Software for Robot Assisted Physical Therapy

Establishing Viability and Prototyping Using Universal Robots UR5

Bjørn Olufsen & Tormod Vaular

Master Thesis at the Institute of Informatics

UNIVERSITY OF OSLO

May 2, 2016
Software for Robot Assisted Physical Therapy

Establishing Viability and Prototyping Using Universal Robots UR5

Bjørn Olufsen & Tormod Vaular

Master Thesis at the Institute of Informatics
University of Oslo
May 2. 2016
© Bjørn Olufsen & Tormod Vaular

2016


Bjørn Olufsen & Tormod Vaular

http://www.duo.uio.no/

Print: Reprosentralen, Universitetet i Oslo

IV
Abstract

The use of robotics in healthcare is increasing. This study investigates the possibilities for moving to cheaper and more mobile solutions for robot assisted physical therapy (RAPT). With software as the main focus, methods for enhanced user experience and added motivation for use is explored. A prototype is developed combining the principles of RAPT, interaction design and gamification.
Acknowledgements

We would like to thank our Supervisors Knut Øvsthus, Geir Omar Berland and Josef Noll for excellent guidance through our work in this thesis.

We would also extend a heartfelt thank you to our friends and co-students Robin and Lars for many great more or less scientific discussions.

Thanks to Emilie for support and allowing us to use the apartment when the school was closed.

Lastly, thanks to the employees at the coffee bar on Kronstad.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Method</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Theory</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Physical rehabilitation</td>
<td>3</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Fugl-Meyer Assessment</td>
<td>3</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Robot assisted rehabilitation</td>
<td>4</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Viability of Robot Assisted Therapy</td>
<td>4</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Approaches to Robotic Assisted Physical Therapy solutions</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Robotic platform</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Programming options</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Physical implementation</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>ROBOTIQ FT 150 Force Sensor</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Robotics</td>
<td>15</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Robotic Joints</td>
<td>17</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Mechanical Chains</td>
<td>19</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Kinematics in Robotics</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>Programming Environment</td>
<td>21</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Windows Presentation Foundation</td>
<td>21</td>
</tr>
<tr>
<td>2.6</td>
<td>Interaction Design</td>
<td>27</td>
</tr>
<tr>
<td>2.6.1</td>
<td>User Experience</td>
<td>28</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Fitts’ Law</td>
<td>34</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Affective Aspects</td>
<td>35</td>
</tr>
<tr>
<td>2.7</td>
<td>Gamification</td>
<td>38</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Advantages with Gamification</td>
<td>40</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Existing Solution</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>Evaluation of the Robotic Platform</td>
<td>42</td>
</tr>
<tr>
<td>3.1</td>
<td>Spatial Accuracy of the Platform</td>
<td>42</td>
</tr>
<tr>
<td>3.2</td>
<td>Testing of Delay and Latency of the Platform</td>
<td>43</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Complete Round-Trip-Time</td>
<td>43</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Stopping Time</td>
<td>47</td>
</tr>
</tbody>
</table>
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example of Fugl-Meyer form. Adapted from [4].</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Graphs showing RAPT compared to TPT. Adapted from [2].</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>UR5 from Universal Robots adapted from universal-robots.com</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Showing the Control Panel Tablet of the UR5.</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>The current handle of the robot. Adapted from [19].</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Showing the UR5 as currently mounted.</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>Specification of FT 150 force sensor. Adapted from manufacturer spec sheet.</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>Showing the FT 150 force sensor mounted on the UR5.</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Showing different degrees of freedom for different joint types [22].</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>Pin joint.</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>Prismatic joint</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>Spherical joint</td>
<td>18</td>
</tr>
<tr>
<td>13</td>
<td>Helical joint</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>Forward Kinematic example.</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>Showing the relations in MVVM. Adapted from [26].</td>
<td>23</td>
</tr>
<tr>
<td>16</td>
<td>Illustrating a data binding. Adapted from [29].</td>
<td>25</td>
</tr>
<tr>
<td>17</td>
<td>Showing usability and user experience goals. Freely Adapted from [30].</td>
<td>30</td>
</tr>
<tr>
<td>18</td>
<td>Fitts' Law. Adapted from [37].</td>
<td>35</td>
</tr>
<tr>
<td>19</td>
<td>Diagram of Fitts' law. Adapted from [37].</td>
<td>35</td>
</tr>
<tr>
<td>20</td>
<td>Screenshots from the PainSquad app. Adapted from [47].</td>
<td>41</td>
</tr>
<tr>
<td>21</td>
<td>Showing Basic Overview of a System Round-Trip.</td>
<td>44</td>
</tr>
<tr>
<td>22</td>
<td>Diagram showing the distribution of results of round-trip-time.</td>
<td>44</td>
</tr>
<tr>
<td>23</td>
<td>Showing measured time in milliseconds in stationary-to-movement tests.</td>
<td>45</td>
</tr>
<tr>
<td>24</td>
<td>Showing relation between time and joint speed of the UR% after an emergency stop. Adapted from manufacturer documentation.</td>
<td>48</td>
</tr>
<tr>
<td>25</td>
<td>Showing stopping distance and time of three of the UR5 joints from maximum speed with 5kg payload. Adapted from manufacturer documentation.</td>
<td>48</td>
</tr>
<tr>
<td>26</td>
<td>Diagram showing distribution of results of stopping test.</td>
<td>49</td>
</tr>
<tr>
<td>27</td>
<td>Diagram showing distribution of stopping times from high velocity movement.</td>
<td>50</td>
</tr>
<tr>
<td>28</td>
<td>Graph showing correlation between force in movement direction and TCP speed, system in servo-mode.</td>
<td>56</td>
</tr>
<tr>
<td>29</td>
<td>Showing simple overview of the system components.</td>
<td>72</td>
</tr>
<tr>
<td>30</td>
<td>Showing the implementation of Point.</td>
<td>75</td>
</tr>
<tr>
<td>31</td>
<td>Showing the URMove class.</td>
<td>76</td>
</tr>
<tr>
<td>32</td>
<td>Showing the classes for exercise and exercise serialization.</td>
<td>77</td>
</tr>
<tr>
<td>33</td>
<td>The tracker class.</td>
<td>77</td>
</tr>
<tr>
<td>34</td>
<td>Showing the ExecuteExercise class.</td>
<td>78</td>
</tr>
<tr>
<td>35</td>
<td>Showing relations between Viewmodels and Views.</td>
<td>86</td>
</tr>
<tr>
<td>36</td>
<td>Showing the initialization chain of mainWindow.</td>
<td>89</td>
</tr>
<tr>
<td>37</td>
<td>Showing individual to total force conversion.</td>
<td>93</td>
</tr>
<tr>
<td>38</td>
<td>Showing relation between force and direction vector.</td>
<td>93</td>
</tr>
</tbody>
</table>
Figure 39 Showing sequence of command updates and their origin................................. 94
Figure 40 Showing translation between force and movement vector.............................. 95
Figure 41 Showing the cone used for deciding sharp direction changes......................... 96
Figure 42 Showing the position tracker with radius on robot approaching point............... 99
Figure 43 Showing robot position after tracker notification and new command sent........... 99
Figure 44 Sequence diagram showing operation of the tracker....................................... 100
Figure 45 Activity diagram showing creation of exercise............................................... 103
Figure 46 Sequence diagram showing creation of exercise............................................. 105
Figure 47 Activity diagram showing execution of exercise............................................. 106
Figure 48 Showing the sequence diagram of executing an exercise................................. 108
Figure 49 Screenshot of the home screen of the application.......................................... 109
Figure 50 Screenshot from the exercise creation screen in the application....................... 110
Figure 51 Screenshot from the exercise selection screen in the application...................... 111
Figure 52 Screenshot from initial state of executing an exercise..................................... 112
Figure 53 Screenshot from calibrating force sensor as second step of executing an exercise......................................................... 112
Figure 54 Screenshot from starting position of executing an exercise.............................. 113
Figure 55 Screenshot showing the force gauge during exercise execution....................... 113
Figure 56 Showing the racing game implemented as gamification................................. 114
### Dictionary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiB</td>
<td>Høgskulen I Bergen (Bergen University College)</td>
</tr>
<tr>
<td>FMA</td>
<td>Fugl-Meyer Assessment</td>
</tr>
<tr>
<td>RAPT</td>
<td>Robot Assisted Physical Therapy</td>
</tr>
<tr>
<td>TPT</td>
<td>Traditional Physical Therapy</td>
</tr>
<tr>
<td>HCI</td>
<td>Human Computer Interaction</td>
</tr>
<tr>
<td>TCP</td>
<td>Tool Center Point (center of tool mounted to a robot)</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transfer Control Protocol / Internet Protocol</td>
</tr>
<tr>
<td>MVVM</td>
<td>Mode-View-Viewmodel</td>
</tr>
<tr>
<td>WPF</td>
<td>Windows Presentation Framework</td>
</tr>
</tbody>
</table>
# 1 Introduction

The continuous progress in technology is bringing forth new areas of use. Within the domain of health care and rehabilitation, these advancements could be especially advantageous as the number of elderly is on the rise, increasing the demand for labor in health care. [1] At the same time, the available work force is decreasing making treatment costlier and less available. Particularly the advancements in computing power and decreased physical size of robotics may have the power to change the landscape of physical therapy in a substantial way. Already a mix of conventional and robot assisted physical therapy is gaining traction and recognition [2], but is still mostly confined to hospital use in cooperation with trained professionals. We would argue that given recent developments within the field of technology, especially robotics, the move to enter patients’ homes is within reach.

The thesis aims to explore the possibilities for developing a prototype system for rehabilitation treatment within the home, as a supplement to traditional personnel assisted therapy. This will potentially give patients additional time of effective training. Health personnel will also be given more time to concentrate on the social aspects of patient interaction, and focus on areas where robotics and computers still cannot contribute in a meaningful way.

In essence, the system developed will consist of a robotic arm, with necessary equipment mounted to it, coupled with a control station in the form of a PC. This architecture is deemed suitable for home deployment, and keeps the complexity to a minimum for easier implementation.

The group of users are seen as patients with movement impairment in upper extremities due to muscular or neurological injuries or defects. Since this set of subjects potentially includes a wide range of demographic groups, the system needs to be designed with this in mind. The development will need to be focused on facilitating an accessible and inviting user experience, including user interface, robotic behavior, and the overall physical implementation.

One of the main challenges in physical therapy and rehabilitation is keeping the patient engaged and motivated. Patient progress depends on amount of training time. Therefore, investigating possibilities for a solution that motivates the patient to be active for the optimal
amount of time is an important goal of the project. To help ensure an increased user engagement the principles of gamification could be incorporated in the system design.

1.1 Method

This master thesis and the research done is made up by five main parts. The resulting prototype has been incrementally developed throughout the process. The five main stages are presented in this report as follows:

- **Study of theoretical background:**
  Central topics discussed and used in the thesis are presented along with previous research in the field.

- **Initial evaluation:**
  Technical evaluation of the platform is conducted, and the viability of choices made in pre development are validated.

- **Implementation:**
  The implementation of the prototype is described and documented.

- **User evaluation:**
  User evaluation of the prototype developed as well as core principles it builds on is conducted.

- **Conclusion:**
  The conclusion of the thesis based on theoretical background, testing, implementation and user evaluation.
2 Theory

This chapter will present the most relevant theoretical subjects used and discussed in this thesis. First the theoretical background for robot assisted physical rehabilitation will be described, before focusing on robotics and software engineering. Finally, psychological aspect of both user experiences and the concept of gamification will be defined.

2.1 Physical rehabilitation

The situations where physical rehabilitation or physical therapy can be of great importance are countless. With patients ranging from stroke victims, trough chronic disease sufferers, to persons with innate conditions. Due to the wide array of possible patients, different programs and treatment plans need to take this into consideration.

Traditionally physical rehabilitation consists of sessions which are overseen and conducted by trained professionals, be it physical therapists, nurses or doctors. These sessions could consist of a number of different activities including motivational talks, planning, education, and both mental and physical exercises. This is a considerable workload and cost, and with the average demographic shifting upwards in age [1], a challenge to supply the demanded workforce needed for the anticipated increase in patients.

2.1.1 Fugl-Meyer Assessment

To be able to measure improvements and effects of physical therapy, be it robot assisted or traditional, a credible metric is needed. Published in 1975 [3], the Fugl-Meyer Assessment (FMA) method gives numerical scores to an individual exercise in a set, and these are cumulated into a total score. The method was assessed in 1993 [4], with the conclusion that the Fugl-Meyer method and accompanying test, is easy to administer, and shows high reliability of the test components, with one exception of pain. The pain evaluation was found to have too big possibilities of error when an absolute scale is used.
2.1.2 Robot assisted rehabilitation

To aid the problem of increasing costs of physical therapy, one solution offered is robot assisted physical therapy (RAPT). One of the biggest advantages with RAPT is potentially drastically reducing the amount of man hours going into each rehabilitation process. By handing repetitive, and the mainly mechanical tasks over to robotics, a lot of labor is freed up for other work. By freeing up time, the health professionals can focus more on the patients and the human aspects. For RAPT to be a viable option or supplement to traditional physical therapy (TPT), the results of such treatment needs to be comparable to the results of TPT.

2.1.3 Viability of Robot Assisted Therapy

In establishing the viability of RAPT as a concept, we will look at two central factors for success which recurs in studies of RAPT. The increase in movement range, and strength will be viewed as deciding factors for the conclusion on the viability of the concept, basing it on the FMA method.

In a study conducted in 2002 [2], two groups of stroke victims were put through rehabilitation consisting of 24 one-hour sessions over the course of two months. One group received TPT,
while the other received RAPT. The study’s aim was to determine whether RAPT used for upper limb rehabilitation could give satisfactory results when compared to TPT. By using acknowledged measures for rehabilitation (FMA), Fugl-Meyer points, strength, and increase in reach extent, a comparison of the two different methods were made.

Looking at the results of the experiments conducted in the study, the concept of RAPT seems promising. A trend of superior results from the group undergoing RAPT compared to the TPT-group can be seen in Figure 2. The study concludes that further research into, in their specific case, treatment of stroke victims with RAPT, is warranted.

In another study of RAPT treatment on stroke patients in 2004 [5], thirty five patients who suffered strokes less than one week prior to the experiments were included. After randomly assigning the subject into two groups, one group started a treatment plan consisting of 4 hours of training with robotic aids per week, for 5 weeks. The control group received robotic training on their unimpaired limb, for 15 minutes twice a week, for 5 weeks. This study relied
on the FMA for result assessment. With results showing substantial improvements in recovery of movement impairments and motor function, the authors conclude that using RAPT in addition to PTP is an effective compliment in the treatment of patients suffering from stroke, even early in the rehabilitation.

In a scientific review published in 2011 [6], several earlier studies were reviewed and controlled, while presenting their own work on a new clinical trial. Comparing results from the different studies, the authors conclude that RAPT can be used as a substitute of other TPT, or in addition to TPT. Pointing to some results showing the greatest improvements in cases where both RAPT and TPT are used in combination.

2.1.4 Approaches to Robotic Assisted Physical Therapy solutions

There is a lot of research on the area of robot assisted physical therapy and in particular concerning human-robot interfaces and control schemes. A number of different configurations of sensors and actuators as well as different mechanical designs has been tested, many of which show great promise [7] [8] [9] [10] [11].

Robots used for physical therapy are sometimes divided into three different classes [8]. The first one being passive robots. Passive robots can only exert resistance to the movement of the patient. The second class is active robots. Through a control system for actuators, these can assist the patient’s voluntary movement whilst making sure not to perform movements harmful to the joints of a patient. The last are hybrid robots which are a combination of the two. Other classifications are also in use [12] with robotics use in rehabilitation. Coaching robots are robots neither aiding or resisting the movement, but are able to track the movement for analysis.

Robots can also be categorized by its structural design, most notably whether it is wearable or not. A wearable robot is known as an exoskeleton [10] [8], and is the robotic counterpart of an orthosis. Orthosis is a device that is designed to fit to the body in order to help achieve certain goals [13]. This may include increasing mobility, reducing pain, controlling biomechanical alignment, and protecting and supporting an injury. It follows from this that exoskeletons in this context are robots that can be worn on limbs or the entire body facilitating and supporting the movements of the user. Exoskeletons should as best as it can try to mimic the intended motion of the user.
The other main class of structural design in robots focused on here are end-point effectors. This design differs in that it is not worn as an exoskeleton is. As the name suggests, the end-point actuators are connected to the subject at the end point, and from there aids or resists movements. This design is simpler, and control algorithms are less advanced. However, the ability to isolate specific movements is less than that of exoskeletal devices.

**Signal Input**

In order to mimic the intended motion of the user, a robotic assistive device will have to properly acquire input signals from the user. There are several types of signals that can be used to control or adjust the motion of a robotic device. The first is biomechanical input where force sensors measure differences in force magnitude and direction to calculate the intentions of a user movement and move accordingly [10]. Some more advanced and intrusive options include connecting to the peripheral nervous system, connecting to nerves in the extremities, or to the central nervous system. The latter mining reading signals directly from the brain through brain computer interfacing (BCI) [14].

Examples of this are electroencephalography (EEG) [15] and functional magnetic resonance imaging (fMRI) [16]. Using these techniques can be very helpful to patients lacking the ability to move voluntarily. It is very efficient at taking advantage of the brains neuroplasticity, which is the ability for a part of the brain to change functionality and take over the task of another part of the brain that has been broken or damaged [9] [7] [17]. This does however require experts to properly tune the device to make sure the signal corresponding to the intention of the user is picked up and interpreted correctly.

Another approach that is less intrusive is using electromyography or EMG [10] signals. EMG signals are generated by the muscle fibers as they contract and relax and can be measured over the surface of the skin. This means the user only needs simple patches with sensors placed over the different muscle groups used to activate robotic device. EMG signals has an advantage over biomechanical signals in that it does not require any force to be applied by the patient. Activity in the muscle is enough to register with the sensors, thus it can potentially better anticipate user intentions.

Most implementations of EMG signals in exoskeletons use an on/off approach in which if a certain criterion is met, a predefined motion is executed [10]. However, this digital switch
does not offer the one-to-one motion synergy you would ideally want between user and machine. It is however suitable for simple motion. A lot of research is focusing on developing algorithms for extracting the intended motions of the EMG signals. Researchers have moved from using input from each EMG sensor individually, into combining data from several sensors and using pattern analysis in feature space in order to find the best interpretation of the user’s signals. Using several sensors gives additional dimensionality to the data allowing for a much more precise interpretation.

A more basic, intuitive and cost effective approach to signal input, is biomechanical input. This is the approach this thesis predominately centers around. Biomechanical input is here used to describe force being applied to the sensors by a human subject. This can either be in the form of a pressure towards the sensor or an alleviation of the downwards or outwards pressure on the sensor. The recorded change in pressure on the sensor can be used to move the robotic device in the direction of the intended motion according to the direction of the change in pressure.

**Some existing solutions**

Many different solutions for RAPT have already been developed. These span all the different classes mentioned earlier, and are spread over a vast selection of use cases and scenarios. Since this thesis will concentrate mainly around upper limb rehabilitation, a few solutions within that area will be reviewed based on previous research gathered. A survey published in 2014 [12] investigates a range of different upper limb rehabilitation robots to evaluate the current environment. It concludes that although many solutions are proposed or already exist, there is a lack in lower cost devices, suitable for potential home installation.

The rehabilitation device for improving hand grasp developed by Park, Jeong, Kwon, Kim and Kim [11], is an example of using biomechanical sensory feedback in physical therapy. They made a robotic device using a pressure sensor on the handle measuring user movement intentions and an infrared sensor measuring the distance from the hand to the sensor. This could be used to measure how far in to the motion the user is at any point while using the device. The device has three different modes of use. A passive mode where the hand is fixed to the handle by a strap giving a continuous pressure to the sensor. In this mode the device will simply help the user do the full gripping motion whether or not the user is able to contribute to the motion. The second mode is an active mode where the device will only help
as long as it detects a pressure from the user. If the user is only able to add pressure halfway through the motion the user will only be assisted halfway through the motion. The third mode is a patient driven (active assisted) mode where the user has to provide a force in order to start the gripping motion but where the device will finish the motion for the user even if the user is not capable of providing pressure throughout the motion. [11]

**Current Challenges**

As mentioned earlier, a lot of the proposed and existing solutions for RAPT are expensive, big and relatively complicated [12]. This is deemed as the biggest disadvantage of using robotics in rehabilitation. To successfully administer treatment to an increasing number of patients, the availability of the equipment and personnel needs to be adequate.

Most obvious is the problem of size and mobility. Complete installations of potentially several hundred kilos and considerable installation procedures are not ideal. To aid this, the complete solutions developed for this purpose should be relatively small and mobile. By doing so, the number of possible treatment venues increase dramatically. If local health centers, doctor offices or even private homes could be used as an arena for RAPT, the amount of patients potentially benefitting from it could increase dramatically.

In addition to physical size, the cost of RAPT solutions is a concern. Smaller and mobile systems could be developed, and new venues for treatment opened up, but if the costs of the system are too high, the spread will still struggle to reach the goal. A possible solution to this is to make use of already mass produced equipment to reduce both developing costs, and the investments and costs of production.

Another aspect of the costs, are the human and labor costs. Given complex systems with lengthy installations, steep learning curves and the need of supervising professionals, the plausibility of the program decreases. If the solutions developed are kept close to “plug-and-play” principles, reducing complexity, they will have a greater chance of succeeding. Day-to-day operation of the equipment should also be possible to do without supervision or guidance, after a training period.
2.2 Robotic platform

The robotic arm used in this project is the UR5 [18]. The UR5 is the midrange robotic arm of Universal Robots with a lift capacity of 5kg, a reach of 850 mm and a total weight of 18.4kg. The UR5 consists mainly of six rotating joints and two aluminum pipes, which connects the base of the robot with the tool of the robot. Each joint has a 360-degree angle of rotation in each direction. The robot has a maximum working speed of 1 m/s and a repeatability down to 0.1 mm, which should be sufficient for the tasks we want the robot arm to be performing.

The robot is being marketed as a very safe and easy to use device, suited for a wide variety of tasks and highly suited for working in close proximity to humans. The robot has several safety features. Most notably, an emergency stop feature which completely stops the robot if it is being subjected to more than 5kg of force. This feature is particularly of interest, as it is the intention that humans will actively be interacting with the robot. and we are going to be interacting with the robot ourselves during testing. A main part of the project will be to make sure that the robot is safe. Users must feel confident using it, and scared that they might get physically injured or have any other uncomfortable experience during interaction.
The tool of the robot is made so that it is easy to attach additional equipment. There is an input plug where sensors can be connected in order to feed information to the robot. The robot can also be programmed to accommodate for the additional payload so that the motions and positions are accurately calculated and the safety features are preserved with the additional equipment attached.

![Figure 4 Showing the Control Panel Tablet of the UR5.](image)

The robot also comes with a touch panel interface for controlling and adjusting the settings, as well as a server station for central processing and external interfacing.

### 2.2.1 Programming options

There are three main ways to interact with, program and control the behavior of the robot:

- **URScripts** are proprietary scripts executed directly from the supplied control panel, or uploaded to a server running in the robot control box.

- Sending commands over TCP/IP to control the robot.

- Writing custom C controller to run on the robot.
One way of doing this is through the control interface that is supplied with the robot. This is a quick and easy way of setting up the robot to perform simple motions and tasks. The interface allows for setting up routines, adjusting the types of motions and setting up security boundaries, which the robot is not allowed to cross. These programs are stored on the robot as URScripts, a proprietary scripting language made by Universal Robots. It also allows for programming movement through moving the robot manually and recording its positions. It is however not very suitable when it comes to continuously changing the motion of the robot based on sensory feedback, due to the relative simplicity of these programs.

A more flexible approach is using the same script API provided by Universal Robots, but transmitting them through a TCP/IP connection to the robot. Using a TCP/IP connection allows for more dynamically changing the motion of the robot based on sensory feedback, by collecting and processing all data on an external computer and generating and sending new scripts to the robot. The UR5 supports asynchronous communication through TCP/IP. Hence, different connections for different tasks can be achieved by connecting to the different ports on the robot. One port can be used for gathering data from the sensors, and another to change or stop the motion of the robot by uploading new scripts. The option of sending commands over TCP/IP is in essence a special case of uploading scripts to the robot. Instead of uploading complete scripts, one-line commands can be sent one by one.

A third option for programming and interacting with the robot is through the C-API. This is a direct “to the metal” programming API which allows for the most precise and efficient communication with the robot. The C-API allows for even more flexibility and control of the robot arm and also potentially a quicker response time and a more efficient code execution. For tasks requiring the quickest response from sensory input to change in the direction or rate of motion of the robot, this API may be ideal. A downside of using this API is it is more complex to handle and the behavior of the robot may change compared to using the script. This is because you are effectively bypassing layers in the robot software architecture and have to do a lot of things yourself instead of relying on high level preprogrammed functionality. Things that are otherwise handled by the upper software layers you might have to do manually through the C-API.
2.2.2 Physical implementation

In order to make the robotic arm an effective tool for use in physical therapy and a practical tool for us to work with, some customization has been done in an earlier project [19]. Currently the robotic arm is mounted on a 75 cm high table with wheels allowing it to be moved effortlessly from place to place. The height of the table makes it suitable for users to stand or sit by the table and comfortably be able to perform exercises. Attached to the tool of the robotic arm is a simple grip (Figure 1) that users can hold when performing exercises and a force sensor that registers the forces that the tool is being subjected to.

The simple design of the handle might be an issue for some patients because it requires them to have an adequate level of force and range of motion in their grip. Given that this system is aimed towards a wide demographic of patients with some potentially suffering from severe movement impairment, this might not be a sufficient design.

While the UR5 has its own internal force sensor, previous projects [19] conducted by the engineering department at HiB has shown that the internal sensor does not have the sufficient precision and reliability required. The internal sensor system in the robot is based on measuring the electrical power in the different joints. This does not give steady and reliable measurements, due to what is believed to be friction in the joints [19]. For this reason, the engineering department has bought an external force sensor, the ft150, that can be mounted on the tool of the robot. Testing verify [19] that it is much more accurate and precise than the UR5 internal counterpart and should be sufficient for the tasks we are trying to accomplish in this project.
2.3 ROBOTIQ FT 150 Force Sensor

As an extension to the UR5 robot, and additional force sensor is added. The sensor can be integrated and run through installing a driver either on the UR5 or on a PC.

**SIGNAL SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measuring range</td>
<td>Fx, Fy, Fz, Mx, My, Mz</td>
<td>±150 N, ±15 N·m</td>
</tr>
<tr>
<td>Effective resolution</td>
<td>Fx, Fy, Fz, Mx, My, Mz</td>
<td>0.2 N, 0.02 N·m</td>
</tr>
<tr>
<td>Signal noise</td>
<td>Fx, Fy, Fz (combined), Mx, My, Mz (combined)</td>
<td>0.5 N, 0.03 N·m</td>
</tr>
<tr>
<td>External noise sensitivity</td>
<td>All axes</td>
<td>Immune</td>
</tr>
<tr>
<td>Cross-talking</td>
<td>All axes</td>
<td>None</td>
</tr>
<tr>
<td>Drift</td>
<td>Fx, Fy, Fz, Mx, My, Mz</td>
<td>±3 N over days, Non-significant</td>
</tr>
<tr>
<td>Data output rate</td>
<td></td>
<td>100 Hz</td>
</tr>
<tr>
<td>Temperature compensation</td>
<td></td>
<td>15°C - 35°C</td>
</tr>
</tbody>
</table>

For operation the sensor needs a power supply of 6-28-volt DC. This is supplied through one of the digital outputs from the robot control box. Data is sent through RS485, RS232 or USB. Data can be read directly through scripts running on the robot, or read through separate programs developed.
2.4 Robotics

There are a variety of ways to relate to and control robotics. Depending on the class of robot, size, capabilities and design, different approaches can be had. Both spatial placement and movement need to be addressed in a satisfactory manner. The way the robotics interact with different entities in the surrounding is also a concern. This chapter will focus on selected parts of robotics related to the chosen implementation.

Placing Robotics in a Physical World

When using robotics there is a need for keeping track of the robotics in relation to the surrounding physical world or environment. This could be done in several different mathematical ways.

The perhaps most intuitive is using a three-dimensional grid representation, often a Cartesian grid, of the space around the robot, and mapping it to coordinates. This gives a straightforward view of the surroundings, and a direct way to instruct the robotics to move relative to a frame of reference. Orienting the frame of reference to different anchors can be beneficial in different use cases. By using for instance the base of the robot, the TCP, or a set point outside the robot as origin of the grid, various problems can be solved easier.
**Robot – Mechanical System**

Robots is often described as mechanical systems [20]. In this sense, we can look at them as linkages, consisting of links and joints. A link in a system is a rigid or flexible body in space. One may think of a link in this fashion as a bone in the human body. In between the links of a system, are joints. Joints interconnect the links and facilitate movement. [21]

**Degrees of Freedom**

In a mechanical system, the term “degrees of freedom” is used to describe or analyze, and is defined as:

> “The number of independent variables (or coordinates) required to completely specify the configuration of the mechanical system.” [20]

To clarify the concept, a geometrical point in two dimensions has two degrees of freedom, given by the x and y coordinates used to specify a point. Moving this point into three dimensional space, an extra degree of freedom is added in the z coordinate.
Further, a rigid body in three dimensional space has six degrees of freedom. Still the same translatory attributes as a point, but now rotation is added by giving rotation around the x, y and z axis. By connecting two rigid bodies in three dimensional space by a pin joint, the system will have seven degrees of freedom. The system still retains the same six as the first body, and adding a pin jointed body rotating in relation to the first body, adds one more degree of freedom.

2.4.1 Robotic Joints

Depending on the robot and the desired, several distinct joints are used. The possibility for different types of joints increases degree freedom for robot movement. Joints are often divided into four separate groups. [20]
- **Pin, revolute or rotary**
  A pin joint is in essence two bodies connected to one axel, allowing for a rotary motion. It can be visualized as the elbow joint in a human arm, or as the hinge of a door. Pin joints have two degrees of freedom since it rotates around one axis.

- **Sliding or prismatic**
  Sliding joints operate with one degree of freedom. The movement is solely along a line. An example of sliding joints is telescopic extendable rods.

- **Ball or spherical**
  A spherical joint is a ball situated inside a compartment allowing for “rolling” the ball. In the human body both shoulder and hip joints are ball joints. These joints have three degrees of freedom.
- **Screw or helical**

Helical joints can be seen as a screw cork. Rotation around the joint moves one body along the length of another. These joints have one degree of freedom since position is solely dependent on the helical angle.

2.4.2 Mechanical Chains

Bodies connected together by joints are seen as chains or kinematic chains.

“A system of rigid bodies connected together by joints. A chain is called closed if it forms a closed loop. A chain that is not closed is called an open chain.” [20]

In addition, an open chain where all links except the first and last are connected to two other links, is called a serial chain. [20] Using mechanical chains to model systems of links can give fairly simple mathematical descriptions of what may seem as complex mechanical systems.

**UR5 as a Serial Mechanical Chain**

By using the concepts described above, the UR5 robot can be broken down to a mathematical model. The robot is made up by seven rigid bodies, one being the mount for the robot, and six joints.

<table>
<thead>
<tr>
<th>JOINT</th>
<th>TYPE</th>
<th>DEG. OF FREEDOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE</td>
<td>Revolute</td>
<td>1</td>
</tr>
<tr>
<td>SHOULDER</td>
<td>Revolute</td>
<td>1</td>
</tr>
<tr>
<td>ELBOW</td>
<td>Revolute</td>
<td>1</td>
</tr>
<tr>
<td>WRIST 1</td>
<td>Revolute</td>
<td>1</td>
</tr>
<tr>
<td>WRIST 2</td>
<td>Revolute</td>
<td>1</td>
</tr>
<tr>
<td>WRIST 3</td>
<td>Revolute</td>
<td>1</td>
</tr>
</tbody>
</table>
To sum up the degrees of freedom we take the three translatory and the three rotational of the first body of the chain, then six more from each of the joints. This gives a total of 12 degrees of freedom.

### 2.4.3 Kinematics in Robotics

Mathematically describing mechanical systems as links, joints and chains gives the possibility to calculate movement and positions of any part of the system through kinematics, the study of motion. This is done to determine the angles of the joints of a robot to set it in a certain pose. A short introduction to forward and inverse kinematics follows.

#### Forward Kinematics

Given a mechanical chain in two dimensions with two links and two joints, we have the specified problem:

*Given the lengths of links l1 and l2, and the angles a1 and a2 of the joints, find the position (X, Y, angle) of the end of l2.*

In a two dimensional space the solution is fairly intuitive.

\[
\begin{align*}
X &= l_1 \cos(a_1) + l_2 \cos(a_1 + a_2) \\
Y &= l_1 \sin(a_1) + l_2 \sin(a_1 + a_2) \\
\text{angle} &= a_1 + a_2
\end{align*}
\]

However, in three dimensional space the solution can be somewhat more complex, but will still be a linear algorithm iterating trough the links and joints to find the final position of the end of the chain.

#### Inverse Kinematics

Perhaps even more crucial to control of robotics, is the inverse kinematics. As the name suggests, inverse kinematics solve the inverse problems of forward kinematics. Given a point in space, solve the angles of all joints in a chain. More practically we can state the problem as,
which angles do the joints \((a_1 \ldots a_n)\) need to be in for the TCP of the robot to be positioned at the desired point.

One big challenge with inverse kinematics is the existence of more than one solution to a given problem. Comparing to a human arm, the index finger can be fixed in one point in space, but there are many different ways of arranging the shoulder, elbow and wrist to still end up at the same point.

### 2.5 Programming Environment

There is an array of available programming environments suitable for developing the solution in this thesis. This chapter will focus on the C#, .NET and WPF technologies from Microsoft, as well as the MVVM principles used in system design.

#### 2.5.1 Windows Presentation Foundation

WPF is a graphical presentation system used for building Windows applications within the .NET framework. At its core, it builds on a vector based rendering engine. It is inherently resolution independent and takes advantage of modern graphics hardware. Interfaces in WPF are created using the Extensible Application Markup Language (XAML). [23]

**XAML**

XAML is the format which is used to define and design user interfaces in WPF. It is a declarative markup language like XML. To separate the runtime logic of the UI from the definition, code behind files tied to the XAML through partial class definitions are used. [24]

Unlike most other markup languages which are typically just an interpreted language, XAML allows for representing direct instantiation of objects through a backing type system.

```xml
<StackPanel Orientation="Vertical" VerticalAlignment="Bottom">
    <Button Style="{DynamicResource RoundedButtonStyle}"/>
    <CheckBox Margin="50,0,0,20" Content="Advance Mode"/>
</StackPanel>
```

In text form, XAML is simply a type of XML file. While defining its own concepts, it is still consistent with the language and markup form of XML. [24]
2.5.2 Model-View-ViewModel

Model-View-ViewModel (MVVM) is a software architectural pattern designed for use in the .NET programming framework. The MVVM design pattern is similar to, and share some of the same concerns as the Model-View-Controller (MVC) and Model-View-Presenter (MVP) design patterns, mainly that of separating the model and the view.

One of the motivations of MVVM as it relates to .NET programming, is to avoid cluttering and complexity in the code-behind file of the view, otherwise known as the user interface. Views made with .NET technologies such as WPF, windows forms, windows phone or Silverlight have a code-behind file that can be used to deal with events in the view and control the UI and presentation logic. When applications get bigger and more complex, so does usually the views. As the complexity and size of the code behind file increase, it can become very hard to manage. Business logic and presentation logic can easily get intertwined and as a result, making changes to the view becomes time consuming and costly. The MVVM design pattern allows for separating and effectively decoupling the presentation logic from the view itself, removing the need for most of the code in the code-behind files. As well as making things tidier, this separation allows for independent swapping of each individual part without making changes to other parts of the system. Changes made to a decoupled part will not propagate to other parts of the software.

Another important point with the decoupling of parts in MVVM is that it allows components to be worked on independently. You can have coders doing the business and presentation logic and UI designers solely focusing on making a great UI without having to worry about the technical details of the business logic.

Microsoft summarizes the main motivations for using MVVM as follows:

1. It provides separation of concerns. Tightly coupled, change resistant, brittle code causes all sorts of long-term maintenance issues that ultimately result in poor customer satisfaction with the delivered software. A clean separation between application logic and the UI will make an application easier to test, maintain, and evolve. It improves code re-use opportunities and enables the developer-designer workflow.
2. It is a natural pattern for XAML platforms. The key enablers of the MVVM pattern are the rich data binding stack of the Silverlight platform, and dependency properties. The combination of these provides the means to connect a UI to a view model.
3. It enables a developer-designer workflow. When the UI XAML is not tightly coupled to the code-behind, it is easy for designers to exercise the freedom they need to be creative and make a good product.
4. It increases application testability. Moving the UI logic to a separate class that can be instantiated independently of a UI technology makes unit testing much easier. [25]

**Structure**

In MVVM the View is responsible for the UI logic, the ViewModel is responsible for the presentation logic, and Model deals with business data and logic. The View is attached to the ViewModel by setting the data context of the View to the desired ViewModel. The ViewModel is attached to the Model by instantiating objects of classes in the Model. Looking at it this way we can say that the View knows which ViewModel it is using, but the ViewModel has no clue what View is using it. In the same way we can say that the ViewModel knows about the Model but the Model has no information about the ViewModel. [25] It should be noted that a single View can use different data contexts for different sections of the View and even for individual attributes in the same UI items. This means a View can in principle have several ViewModels. Conversely, one can also use the same ViewModel in several Views.

**Communication in MVVM**

To help facilitate the decoupling of UI and presentation logic (view and view model), MVVM makes use of data binding, commands and notifications.

Data binding is described in more detail in the next section, but in short it is the linking of two properties such that the value of one is linked to the value of the other. [27] Data binding can go both ways, or only one way. In MVVM it allows for letting properties located outside the view, display directly or change attributes within the view. In the context of MVVM, data binding is used between components in the view, and properties in the view model.
Commands are used in binding an action in the user interface, like for example a button click or an item selection, to a function, specifically an event handler, in the view model. It is essentially the same as binding an attribute except it is not a regular property you are binding to but a command property. The command property contains a function that is executed when the action it is bound to is performed. It also contains a function that determines whether or not the action can be performed through a boolean return value. Like regular data binding, using commands is a good way of reducing code in the code-behind file.

Notifications is the way the view model signals the view that something has changed in the view model that the view needs to reflect. This is usually done through implementation of the INotifyPropertyChanged and INotifyCollectionChanged interfaces, and is using the .NET event system. The notification event is typically triggered in the set function of properties that is binding to the user interface. This way the event will trigger every time the property is changed, telling the view to update with the new value.

**Data Binding**

The purpose of data binding in WPF is to bind a variable or an attribute in the view, also known as the user interface, to a property located outside of the view, such that change in value in either properties causes that same change to occur in the other one (two-way binding). Alternatively change is reflected only when one of the two elements changes value (one-way binding). Data binding is the process that establishes the connection between the different properties. [28]

In WPF data binding fulfills the purpose of connecting the user interface with the business logic such that intermediary steps of getting the information to and from the user interface is limited. Instead of writing code for getting values from a UI element and putting it in a variable, one is effectively working directly with the underlying variables when making changes to a UI element that is bound to that variable. A typical example would be a textbox that is bound to a property in the code. When typing something in the textbox the property is automatically updated, reflecting what was typed in the textbox.
Figure 16 Illustrating a data binding. Adapted from [29]

Data binding can be seen as the bridge between the binding target and the binding source. A data binding typically has the four components illustrated above. The binding target consists of a dependency object, typically a UI element such as a button or a textbox, and the dependency property, which is the specific property that is being bound, for instance the text property of the textbox. The binding source consists of an object and a property of that object. This could be an object of type Patient, and a property called Name, which is binding to the dependency property of the binding target. It is also possible for the dependency property to be bound to the object itself and not an underlying property, in which case the object acts as both the object and the property of the binding source.

Binding Types

There are four different ways of data binding. One-way binding, which is from Source to target. Two-way binding goes both ways. OneWayToSource is from Target to Source, and finally, OneTimeBinding which works like a snapshot of the state at the moment the binding occurs. It does not propagate any changes that occur in either target or source. [28]

Triggering Changes

The binding-target does not automatically detect changes made in the source. A notification will have to be sent telling that the source has been updated in order for the target to reflect that change. This is typically done through implementation of the INotifyPropertyChanged or INotifyCollectionChanged interfaces.

If a binding is either TwoWay or OneWayToSource it will listen for changes in the target property and propagate that change to the source property. In order to have some flexibility with when and how often the source is being updated, the binding target has an UpdateTrigger property. The value of the UpdateTrigger property is the event that determines when the binding source property should be updated. In most cases this property is set to
PropertyChanged in which case the binding object is essentially subscribing to the PropertyChanged event of the target property. However, this property can be set to other values, like LostFocus, in which case it will only update the target property when the LostFocus event has been triggered.

**Specifying the binding source**

In order to create a binding, a binding source must be specified. There are several places and ways to do this. Perhaps the most normal way is to specify the source object through setting the data context and then specifying the specific property to bind to by setting the value of the attribute you want to bind to, to a binding statement as such:

```xml
<Window.DataContext>
    <SomeDataContext/>
</Window.DataContext>
<Grid>
    <TextBox x:Name="textBox" Text="{Binding Path=SomePropertyName}"/>
</Grid>
```

Here, SomeDataContext is set as the data context of a window, and the text property of a text box is bound to a property SomePropertyName that is assumed to exist as a property of the data context object. One can also set the data context individually for each element, or for each group of elements. This means different data contexts can be set for different items, for instance the grid could have its own data context that would then be inherited by every child element of the grid. Alternatively, one could specify the binding source explicitly in the binding statement for the individual element as such:

```xml
<Button Width="150" Height="30"
    Background="{Binding Source={StaticResource myDataSource},
    Path=ColorName}"
    I am bound to be RED!"/>
```

Binding Source refers to the binding source object, and Path refers to the binding property. In some cases, the binding tag is used by itself with no path or source specified. This means the binding path is being set to the source itself, which is whatever the current data context for that section of the interface is set to.
2.6 Interaction Design

Interaction design is a broad multidisciplinary field built around designing interactive products that support the way people communicate and interact in their daily lives. [30] It is used as an umbrella term for a vast array of design terms, such as user interface design, software design, user-centered design, experience design, product design and interactive system design. [30]

Separating interaction design from traditional software engineering is a focus on the user and its interactions, rather than the technical functioning of the program. [30] Interaction design is concerned with functionality as it relates to supporting the user in their everyday interactions. While both software engineering and interaction design is concerned with functionality, the former is concerned with the specific task, while the latter is concerned with the usability and usefulness of the interactions, and with creating a good user experience. While having a different focus, interaction design is not at odds with regular software engineering, and should be seen as a complement rather than a replacement.

Fundamental to the field of Interaction design, is understanding how users and technologies communicate with each other. This allows for anticipating how people might interact with a system, giving the opportunity to fix problems early on in the design process or avoid them entirely, as well as inventing new ways of doing things. [31]

Understanding User

An important concern within interaction design is how to optimize for the user’s interaction with a system or with the environment, such that it enhances and expands the user’s activities in useful and effective ways. [30] In order to design for this in a principled manner, it is important to base design decisions on an understanding of the users. This involves:

- “Taking into account the strengths and weaknesses of the users.
- Considering what might help people with how they currently do things.
- Thinking through what might provide quality user experiences
- Listening to what people want and getting them involved in the design
- Using ‘tried and tested’ user-based techniques during the design process.” [30]
2.6.1 User Experience

User experience is a central concept in interaction design. It is tied to the way a product behaves and how it is used by people in the real world. Every product that is used by people can be said to have a user experience. [32] The user experience is defined by how people feel about a product. What it is like to use it, and what emotions arise when looking at, holding, and using the product. [30] This includes the overall impression as well as small details like the satisfaction of opening or closing the lid on a bottle, or the pleasant sensation from touching the well-rounded surface of an ergonomically designed game controller. It is important to note that one cannot design a user experience, only for a user experience. This is because the user experience is not the product itself but how people experience using the product in the context of their daily lives. [30]

Usability and User Experience Goals

When designing an interactive product, it is important to have a clear idea of what the objective of that product is as it relates to the user. Setting usability and user experience goals is a useful way of identifying purposeful requirements for the product, as well as gaining a better understanding of the user.

Usability Goals

The concern of usability goals is with criteria of usability. Usability is about making sure that an interactive product is easy to learn, effective to use, and enjoyable from the perspective of the user. [33] Usability can be broken down into the following goals:

- “Effective to use (effectiveness)
- Efficient to use (efficiency)
- Safe to use (safety)
- Having good utility (utility)
- Easy to learn (learnability)
- Easy to remember how to use (memorability)” [33]

Effectiveness is referring to how good the product is at doing what it is supposed to do.

Efficiency is referring to how much effort and how many steps are needed to complete the interaction. A more efficient design is quicker and requires fewer steps.
**Safety** is about protecting the user from harm and undesirable situations when interacting with the product. This includes external aspects like potential dangers with the physical product itself or with the work environment the product is used in. For instance, interacting with robotics can involve a risk of physical harm to the user, and in the case of a work environment, hazardous conditions might require a product to be remote controlled. Safety also involves the aspect of avoiding unwanted actions to be carried out. [33] For instance, if you do not want the user to be able to delete the entire company database, do not give them the ability to do so. Included in this is also the perceived fears of the users about the likelihood and severity of making a mistake and how this might affect their behavior. Means of addressing this concern includes careful placement of critical operations and options for recovery if a mistake has been made. [33]

**Utility** is about providing functionality that allows the users to do what they need or want to. It is in part about how much functionality a product provides, but it is also about having the right functionality. In the end, it is the usefulness of the functionality that determines its utility.

**Learnability** refers to the ease of learning a system. People generally do not like spending time learning how to use a system. It is important not to make a system harder to learn than the amount of effort the user is likely to want to put into learning how to use it. Users may to a certain extent be willing to spend a little more time learning a more complex system than one with a smaller scope.

**Memorability** refers to the ease of remembering how to use a system or product, once learned. It is particularly crucial for operations that are not used very often. Having a good logical structure and using meaningful icons, names and menu options adds to the memorability of the system.

**User Experience Goals**

While usability goals are generally concerned with the usability of the product from its own perspective, user experience goals are concerned with what the user experience from their perspective when interacting with the product. [33] Particularly the feelings that the product give rise to in the user. User experience goals include things like how satisfying, enjoyable, engaging, motivating and aesthetically pleasing the product and the interactions are. Some
user experience goals are only partly concerned with the emotional response, like whether or not the product is challenging, or whether or not the product is enhancing sociability. One might very well feel that a product is enhancing sociability, but whether or not it actually does is not determined by that feeling. However, it is still determined from the perspective of the user. It is important to keep in mind that different users can have different user experiences with the same product and while performing the same actions, and as such the experience is a relation between the user and the product.

![Diagram of usability and user experience goals](image-url)

*Figure 17 Showing usability and user experience goals. Freely Adapted from [30]*

**Evaluation of Goals**

While for the most part focused on the subjective nature of the interactions, user experience goals can still be objectively verified. The challenge is to be precise when dealing with the qualitative aspect of the user experience. Usability goals generally lend themselves better to empirical verification, because they can often be broken down into quantitative measurements. For instance, efficiency can be measured by the time it takes to complete a task and learnability by the time it takes to learn and remember how to use a product or a
specific functionality of the product. In order to evaluate the user experience concepts, it is useful to ask specific questions. [33] It is easier to measure and it helps in making a more concrete foundation to base decisions from.

**Interaction Design Process**

In their book, Interaction Design: beyond human-computer interaction, Rogers, Sharp and Preece identifies four basic activities involved in the interaction design process:

1. Identifying needs and establishing requirements for the user experience
2. Developing alternative designs that meet those requirements
3. Building interactive versions of the designs so that they can be communicated and assessed
4. Evaluating what is being built throughout the process and the user experience it offers.

[30] The process is iterative, and each activity is intended to inform one another, ensuring that new information is continuously taken advantage of. [30] Interaction design has a strong emphasis on evaluation of what has been built. Evaluation is usually performed through a user-centered approach that seeks to involve users throughout the design process. Involving users is a great way of finding strengths and weaknesses with the design. It allows for a better understanding of what the user experience of the product is going to be like and how one might go about to improve on it.

**Design Principles**

The field of interaction design has a vast number of design principles with varying degrees of recognition. These principles are intended to help guide designers in making sound decisions for the design of the user experience. The most common principles deal with determining what the users should see and do when performing tasks with an interactive product. [33] The five most common principles according to Rogers, Sharp and Preece are:

- Visibility
- Feedback
- Constraints
- Consistency
- Affordances

[33]
Visibility is about showing the users what options they have. If functionality or information is hidden or poorly visible, it is difficult for the user to know that it is there, or what actions need to be taken to take advantage of it. The more visible the functions are, the more likely the user is to know what to do next. [33]

Feedback is about giving responses on the interactions of the users so that they know that their actions are having an impact. Without feedback, users cannot be sure that they have accomplished what they intended by their actions. Did pushing that button do what I think it did? Feedback can be given both on the action being performed and on what that action has accomplished. [33] This could mean an animation on a button showing that it was pressed, and a message on the screen telling you that the document was saved, the purchase was made, or perhaps that you need to fill in more information. A good use of feedback adds to the visibility of the interaction.

Constraints are about restricting the possible user interactions that can take place at a given moment. The purpose of this is to direct the user towards the functionality that is useful or necessary at that moment, as well as directing the user away from the things that currently does not require their attention. [33] A common practice in graphical user interface design that exemplifies this is to deactivate and shade out certain menu options that should not currently be accessible, thereby avoiding the possibility of making mistakes. Constraints could also be physical, like only having one way to attach a cable. The result is the same; the user is directed towards making the correct actions.

Consistency is about making an interface that follows predictive rules. Similar parts should follow the same or similar rules to the extent it is practical. For instance, selecting items should be performed by the same action across the interface. [33] This applies to any part of the interface. For instance, maintaining a similar look for elements of the same type, and the same look for elements that perform the same action adds to the visual consistency of the interface. One of the benefits of a consistent design is that it is easier to learn and easier to remember how to use the system. Absolute consistency is however not necessarily something you want to strive for. Using the same type of operation for all tasks would make for an operationally consistent design, but would in many cases be very inefficient. For example, an interface where every interaction was done through button presses. It could work fine for a system with only a few functionalities, but increase the amount of possible operations and the interface would quickly become messy and impractical.
Affordance is used to describe how an object invites a certain action to be performed on it. To afford can be seen as giving a clue. For example, a door handle has the affordance that it can be pulled, such as to open the door. One could say a door handle affords pulling, while for instance a button affords pushing. An object can be said to have actual and perceived affordances. An actual affordance lies in the physical nature of the object and what that allows. The circular shape and bounciness of a ball affords rolling or bouncing, in the way that it allows for those actions to be made based on its actual physical properties. Perceived affordances lie in the cognitive interpretation of what actions can be performed on an object. A GUI element for instance, may not have any real affordances, but it can still have perceived affordances based on what it looks like you could do with it. The main purpose of affordances in interaction design is making the interactions intuitive. Creating objects that afford the correct actions makes for an intuitive design and ensures minimal need for explanation through symbols, text or other means.

Nielsen’s 10 Heuristics

Design and development of user interfaces is one of the most crucial elements of a system development. Jakob Nielsen has worked out ten heuristics for usability, which are widely acknowledged. Using these as basis for GUI design and architecture should give a better result.

- **Visibility of system status:**
  The user of a system should at all times be aware of the system status. This is to avoid confusion around if the system is working as it should, something is happening, or if the system has crashed.

- **Match between system and the real world:**
  Using terms and expressions known to the user from the real world, will make the system a lot easier to understand and get accustomed to. Naming a part of a system where you buy and sell merchandise a store or marketplace, will make the user instantly familiar with the concept, and leave further explanation unnecessary.

- **User control and freedom:**
  There should always be room for the user to try actions without being left stranded. Including mechanics to reverse actions without having to redo work will give the user more freedom to do errors, or to just explore to find what they are looking for.

- **Consistency and standards:**
  The words or expressions should have the same, consistent meaning throughout the
system to avoid confusion.

- **Error prevention:**
  Attempting to design systems such as to avoid error-prone areas could greatly reduce the amount of errors users commit. Additionally, prompting confirmation from the user before executing crucial actions could reduce number of errors.

- **Recognition rather than recall:**
  Having options and choices visible to the user, rather than the user having to enter them from memory will make the system more user friendly by taking a lot of the work off of the user.

- **Flexibility and efficiency of use:**
  Adding ability for advanced and expert users to use shortcuts and hotkeys to navigate the system more quickly, while still having the regular guided route for novice users, makes the system more flexible towards a mixed user base.

- **Aesthetics and minimalist design:**
  Presenting the user with relevant information only, increases the chance of successful communication. In addition, removing unnecessary clutter from the system makes it more clear and straightforward.

- **Help users recognize, diagnose, and recover from errors:**
  Presenting users understandable error messages increases the chance of the user identifying mistakes, learning from them, or even rectifying them immediately. Furthermore, it may help to suggest a solution to the user through the error message.

- **Help and documentation:**
  Supplying a user manual or documentation with the system gives users a resource for support on using the system. Having the possibility of getting help in context within the system itself could give users a way to find the next step if they become lost.

These are principles to keep the user oriented and more easily traverse the system.

### 2.6.2 Fitts’ Law

Fitts’ Law is an empirical model that is concerned with the speed-accuracy tradeoff characteristics of human muscle movement. It was created by Paul Fitts in the 1950’s as an extension of Woodworth’s research on telegraph operator performance. Although originally targeted at worker efficiency problems of that time, such as production lines, the advent of the graphical user interface has ensured that Fitts’ law remains relevant today. [37]
The parameters of interest in Fitts’ law are the time it takes to move to the target, the distance between the starting point and the target center, and the width of the target. Fitts’ law is as follows:

\[ MT = a + b \log_2(2A/W) \]

*Figure 18 Fitts’ Law. Adapted from [37]*

In this equation, MT represents movement time, A stands for amplitude which is the distance from starting point to target center, W is the width of the target and a and b are constants that have been determined by empirical research. [37] Increasing the size of the target or decreasing the amplitude are the contributing factors in reducing the movement time to the target.

*Figure 19 Diagram of Fitts’ law. Adapted from [37]*

The main lesson taken from Fitts’ law as it relates to interaction design, and human computer interaction in particular, is that the placement and size of elements are directly related to the time it takes to perform tasks in a graphical user interface, and can be an important factor in the overall efficiency of the system.

### 2.6.3 Affective Aspects

Traditionally much of the focus in designing interfaces has been on the practical functionality and usability of an interface. Emotional aspects have generally been thought of in the sense that a usable product is satisfying to use and a poorly designed product is potentially quite frustrating. Emotional response has for the most part been considered as a byproduct of good
or bad design as opposed to something to design for. Increasingly emotion, and affective aspects, meaning aspects that generate an emotional response, is becoming important. [38]

Humans are generally very good at expressing and recognizing emotions. They have a tendency of reflecting the emotions of others. When a person is being smiled to, that person is likely to smile back. It is not only that the emotional expression is being reflected, but it is also affecting the emotional state of that person.

There are a number of ways humans can be affected by an interface. Virtually anything in an interface can be designed to have affective properties. Affect can be induced through use of icons and animations, expressive avatars and agents, through sounds, and through the use of different colors and images. An aspect that has been getting increased attention in human-computer interaction (HCI) is aesthetics. HCI used to be focused on usability and not aesthetics, but empirical research has shown that the aesthetics of an interface positively impacts the perception people have about its usability. [39] Users are likely to be more tolerant of an interface that is elegant and pleasing to the eye, and it is likely to increase their sense of satisfaction and pleasure when interacting with it.

While it is possible to enhance the experience of the user through affective design aspects, it is also possible to do the opposite. Aspects that are intended to induce pleasure or reassurance to the user can end up as frustrating if it is hampering the usability of the product. Use of so called friendly agents or of cutesy looking avatars might be fine at first, but could quickly become annoying. Particularly in tasks that are performed often. [38]

A way to take advantage of affect is trough persuasive techniques. It is about using affect in such a way as to make people do things or to change their mind. An example of this is Nintendo’s Pocket Pikachu which aided by an inbuilt pedometer, requires the user to exercise regularly in order to keep the Pikachu alive. A technique that is often used for persuasion is the theory of positive reinforcement which states that an activity is more likely to be repeated when a reward is occasionally and randomly given. [38] This technique is used by for instance slot machines. Many useful techniques that can be used in persuasion come from game design. These can be put to use through the concept of gamification which entails using design principles from game design about how to make something fun, engaging and motivating, and applying it to tasks that are usually not considered to have those qualities.
This has the potential of increasing the time spent doing those tasks as well as making it a more pleasant experience.

**Anthropomorphism**

Anthropomorphism is the case of giving non-human or inanimate objects human qualities. Humans and children in particular have a tendency to like objects that has been given human qualities. [38] It is argued that human attributes generally make things more enjoyable and motivating to interact with. For instance, addressing a user in first person is more appealing then impersonal third person and can make them feel more at ease, reducing stress and anxiety levels. [38]

Critics of the anthropomorphic approach argue that they can have the opposite effect, making people feel anxious and inferior as a result of it. For instance, when making mistakes; being addressed in first person can be harder than an impersonal third person response, if the message can be interpreted in a negative light. Shneiderman argues that anthropomorphic interfaces, particularly those using first person dialog and screen characters are deceptive. [40] The most common issue when giving human attributes to a computer is that it can really quickly become annoying if it does not meet the expectations of the user. Making an interface more human like raises the expectations the user has about its intelligence and capabilities. They are even expected to have personalities and emotions. Not meeting these expectations can result in the user becoming distrusting. [38] Still, studies have shown that anthropomorphism can be used for positive effect. A study by Walker et al. showed that schoolkids would spend more time with a display containing a talking head than an equivalent display showing only text, and subsequently score higher on a test. [41] A follow up study conducted by some of the same authors, showed that users would present themselves in a more positive light, and interact more with the talking head display. [42] On the other hand, it caused some of the users to be disconcerted or displeased, arguably due to the stern facial expression of the talking head. [38]

**Emotions’ Effects on Cognition**

Emotions change the way people think. When frightened or angry, thoughts become narrow and focused, with purpose of dealing with the source and potential danger that is causing these emotions to arise. On the other hand, emotions of happiness and satisfaction cause the
body and mind to relax, and is believed to allow for wider and more creative thinking. [38] Negative emotions like depression and anxiety can affect the decision-making abilities of people by narrowing their perception of available options. They are also shown to impact logical reasoning. People under the influence of positive affect score significantly better in this aspect than those influenced by negative affect. However, the highest scores seem to be achieved when under the influence of neutral affect. [43]

2.7 Gamification

Opinions differ on when the term was first used, but the term seems to surface a few years after the millennium. Towards the end of 2010 gamification entered into the mainstream domain, reflected in google search statistics.¹

Several authors have proposed a definition for gamification since the term appeared. The most widespread and adopted definition² seems to be from “From Game Design to Gamefulness: Defining ‘Gamification’” [44], and this will form the basis for the understanding of the word further on in this thesis.

“To summarize, gamification refers to

- The use (rather than the extension) of
design (rather than game-based technology or other game-related practices)
elements (rather than full-fledged games)
characteristic for games (rather than play or playfulness)
in non-game contexts (regardless of specific usage intentions, contexts, or media of implementation).” [44]

As Deterding notes, gamification is spreading to a lot of different domains; finance, learning, health, to name a few. The shared goal of these is increasing user engagement and motivation to encourage more usage of the system. Numerous approaches and what Deterding calls “game elements” are used in the gamification. On the relatively simple end of the scale are elements like points, levels and leaderboard, familiar to most from games in all genres and degrees of sophistication. The addition of this as a layer on top of an existing system, or as

¹ http://www.google.com/trends/explore#q=Gamification
part of a new development is fairly straightforward. However, gamification is not limited to these elements.

To define what a game element is, we look to games, designed as games. Using Reeves and Read [45] list “Ten Ingredients of Great Games” as a basis for what might be, and not a complete set of, such game elements, we can discuss some different implementations.

- **Self-representation with avatars:** Representing users with avatars could give a sense of ownership while making the user invested.
- **Three-dimensional environments:** Building a three-dimensional environment to interact with or explore will help envelop users further in the game-setting.
- **Narrative context:** Adding a narrative to the game will make progression through it more enticing and keep user interest at a higher level. This could be a set storyline, or one which changes depending on user choices and actions.
- **Feedback:** Giving users what is known as positive feedback in games is giving boost, items or better attributes as rewards for objectives completed, which then in term makes the user better suited for more difficult objectives. This gives an additional incentive for users to complete objectives, and rewards skilled users.
- **Reputations, ranks and levels:** Giving users the ability to gain ranks, levels or earn reputation in a game is one of the most direct ways of progression. New levels or ranks will give users clear achievable goals to reach and constantly supply objectives to fulfill.
- **Marketplaces and economies:** Introducing economy and markets to a game will add another dimension to the game-world. Currency and goods gives additional incentives for users to spend time in the game.
- **Competition under rules that are explicit and enforced:** Presenting users with competitions bound by clear, enforceable rules may trigger the natural competitiveness in them, increasing their level of engagement to the game.
- **Teams:** The possibility of teams can both add a social aspect to the game, as well as making harder objectives obtainable with cooperation.
- **Parallel communication systems that can be easily configured:** Having the possibility of an in-game chat or voice-chat, and ability to create specified channels in these, will give users the ability to communicate with other users, groups or the game society at large as they please.
- **Time pressure:** Putting a time limit on certain objectives in a game, can have the effect of engaging users additionally. This might also give a greater sense of achievement for the user when completing the task.
2.7.1 Advantages with Gamification

As previously stated, the main goal of gamifying a system, is to increase user engagement and motivation in a given environment. By applying additional objectives, or masking the original objectives with new ones, the user may be compelled to put in extra effort. This can be especially beneficial when the initial tasks or objectives seem tedious, repetitive or even boring to the user.

One of the big benefits with using games as a motivator, is the ability to tailor games to different user groups and demographics. The way to engage a toddler may vary widely from that of elderly users. Additionally, the ability to add or remove complexity to games by use of levels or difficulty settings, can ensure a more including solution. This is at the core of gamification. By regularly changing or updating game modules one may also keep a user engaged for longer periods of time.

Gamification has been shown to have the ability to positively emotionally enhance the user experience, for instance optimism, encouraging curiosity and pride. [46] The concept may even in some cases negate negative experiences, and potentially turn them into positive ones.

These potential effect of gamification could be especially beneficial in health, and particularly rehabilitation applications. Within this area one of the biggest challenges is precisely giving the user positive experiences, in an inherently negative setting.

2.7.2 Existing Solution

As mentioned gamification has been implemented in a lot of different environments. The perhaps most known and tested environment is within learning. This could be either a math game for children used in schools, or the increasingly popular e-learning solutions used for employee training. A lot of valuable research done on these can be transferred to other domains, though there is far from a complete overlap.

In the healthcare domain there is an increasing number of solutions drawing on the benefits of gamification. The Hospital for Sick Children, Sickkids, a hospital affiliated with the University of Toronto, Canada, have introduced a mobile application for use in the treatment of cancer patients. [47]
To correctly assess how to further manage their patients' pain, doctors use pain journals to keep track of pain levels, and getting sick children to do this has been a challenge [48]. This is the reason for the Pain Squad™ mobile application. In the app, children get recruited to a special police force, the Pain Squad, and are asked to perform certain tasks. The objectives given to the user are centered around helping the pain squad track and catch “pain”, the villain of the game. Through missions, the children answer questions about, and gauge their pain. Completing these tasks give the children various rewards in the game – giving them a bigger incentive to complete them, while giving the health personnel invaluable information in the form of a digital pain journal.

Figure 20 Screenshots from the PainSquad app. Adapted from [47].

Cundari, Sickkids development partner for the Pain Squad™ app, reports that compliance numbers in keeping pain journals went from 11% to 81% with the app compared to a traditional pen and paper journal [48]. This shows how use of gamification practices can improve user engagement where motivation might be a challenge.
3 Evaluation of the Robotic Platform

This chapter will aim to evaluate the technical capabilities of the platform chosen for the thesis. To aid evaluation, a set of tests will to be conducted. Since the robot will be working directly with human limbs, safety is paramount, and certain abilities and attributes of the platform will have to be established and confirmed. This will be addressed through three main analysis of spatial accuracy, delay and latency, as well as an evaluation of use of the force sensor. In addition to these specific subjects, a more general proof of concept will also be shown to validate the platform and choices made in pre-development.

3.1 Spatial Accuracy of the Platform

The first subject for technical evaluation of the platform is the spatial accuracy. By spatial accuracy, both direct accuracy and repeatability is addressed. Although further testing could be conducted, the documentation from the manufacturer as well as observations done in development will be laid ground for the evaluation of this.

Data

The spatial repeatability of the UR5 platform is given from the product documentation to be +/- 0.1mm. Further, observations done in development showed the output containing position data would fluctuate up to about 0.1mm.

Discussion

Several aspect of the spatial accuracy evaluation could be improved. Most importantly, independent testing should ideally be conducted. However, it was decided that in the scope of this thesis, other subjects were more central. Further testing would require acquisition of specialized equipment, and is seen to not add sufficiently improved data. Specifications given in the documentation is deemed to be well within any threshold we see necessary for use with biomechanical movement. Use of the robot, both by the authors and by others at the department, have also shown the spatial accuracy to be satisfactory. Even when used for task needing higher precision than demanded for the use in this project. It has therefore been decided that any extensive testing of the spatial accuracy, as well as repeatability of the robot, is not needed.
Conclusion

It is concluded, given the data and the assumptions described in the discussion, that the spatial accuracy of the platform is satisfactory for the task. No actual direct accuracy is tested or discussed, and only data on repeatability is available. However, in the scope of the thesis, there is no strong need for accurate placement other than repeating set movements.

3.2 Testing of Delay and Latency of the Platform

The next subject to be evaluated is the delay in and the latency of the platform. Several slightly different experiments are conducted to examine time limitations of the robot alone, and the system as a whole.

3.2.1 Complete Round-Trip-Time

The chosen robotic platform does contain certain safety measures in itself, including an emergency stop function, but the ability to stop or change a movement from the control application will still be important. It is therefore a need to measure the time it takes from change is detected, through issuing a new command, until the new command can be detected as change in the robot again. This is referred to as a “round-trip” in the system.

The round-trip-time is tested and established through a fairly simple constructed experiment. First the robot is set to a certain position. From the control application on the connected PC, a new movement command is then sent. Along with sending the command, a timestamp is saved in the application itself. The application continuously checks the input from the “real-time” interface, and as soon as movement is discovered, a second timestamp is saved. The two timestamps can then be compared, and the total time from sending command to receiving data about the command being executed can be calculated. The test is run several times to check if the result is fairly constant, or to what degree it varies.
Results from Testing Delay and Latency

The testing of round-trip-time were set up with different movement paths, speed and acceleration values to give a broader result base for analysis. Complete data can be seen in the appendix. The time measurements of the stationary-to-moving tests are shown below.
The graph above shows the distribution of the results from the stationary-to-moving test. A total of 40 runs of the robot were conducted.

To illustrate a possible trend in the experiment, all measurements are shown in the graph above. The measurements are done in groups of ten movements between the same two Cartesian points in alternating direction. The second group in particular seems to suggest that direction can be a contributing factor to total time delay and latency of the system.

The values for maximum and minimum time elapsed between command issued, and movement detected:

- **Maximum time**: 1828 milliseconds
- **Minimum time**: 909 milliseconds

The average of the entire experiment, including different values for velocity and acceleration is:

- **Average time**: 1399 milliseconds
Other general results of interest:

**Median:** 1416 milliseconds  
**Standard deviation:** 251 milliseconds

As stated, the experiments were divided into different movements, acceleration values and velocities. Examining the differences between the groups we find some variations across.

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Exp1</th>
<th>Exp2</th>
<th>Exp3</th>
<th>Exp4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>1399</td>
<td>1367</td>
<td>1227</td>
<td>1724</td>
<td>1280</td>
</tr>
<tr>
<td>Max.</td>
<td>1828</td>
<td>1452</td>
<td>1517</td>
<td>1828</td>
<td>1586</td>
</tr>
<tr>
<td>Min.</td>
<td>909</td>
<td>1261</td>
<td>953</td>
<td>1828</td>
<td>909</td>
</tr>
<tr>
<td>Median</td>
<td>1416</td>
<td>1368</td>
<td>1231</td>
<td>1722</td>
<td>1390</td>
</tr>
<tr>
<td>Std.Dev</td>
<td>251</td>
<td>54</td>
<td>225</td>
<td>64</td>
<td>210</td>
</tr>
<tr>
<td>Variance</td>
<td>62970</td>
<td>2945</td>
<td>50490</td>
<td>4051</td>
<td>44171</td>
</tr>
</tbody>
</table>

All values are given in milliseconds.

**Analysis of the Stationary-to-Moving Tests**

From the test conducted in different test scenarios, a few general points are of notice. Firstly, it is apparent that there is some deviation as far as delay and reaction time of the robot is concerned. Although there seems to be a certain correlation between the position of the robot, the direction of the movement, and the time elapsed before change is registered, there are too many factors to draw a definite conclusion.

Since measured position is done through internal sensors in the robot, and published through one of the servers, there are a big degree of uncertainty to consider. Initially, the values for current position retrieved through the ‘real-time’ server fluctuate somewhat even when the robot is in a stationary and locked position. This gives rise to the first error source. To handle these fluctuations, a certain degree of rounding is done to get stable measurements. In practice, this leads to movement being detected only after a sufficiently big change in position has occurred, potentially adding significant time to the measurement.

Further sources of errors are also present. The time measurement itself is based on an internal timestamp function embedded in the Windows operating system. Since the OS is not operating in real-time, there could potentially be further addition to the actual time here.

When taking all the different error sources into consideration, it is not possible to draw too bold conclusions from these tests. However, some clear indications become apparent, and
some useful knowledge can be acquired. The round-trip time when sending a movement command can be anticipated to be between one, and one and a half seconds.

Although there are imperfections in the tests conducted, the results are useful for further development. The fact that the robot might be moving before it can be measured, or that some time delay can come from sources other than the robot itself, lessens the value of the results for evaluating the robot isolated. However, the system as a whole will suffer the same sources of error in normal operations, and the tests will in that sense mimic reality quite closely. The times measured will represent times from commands are sent until new calculations can be made on the updated position of the robot.

**Conclusion on Stationary-to-Moving**

The results from testing does show quite a big variance in time before movement is detected. This does give rise to some concern around weather the current approach to dynamically controlling the robot from a pc application will be able to accomplish smooth and reactive movements. However, there are clear limitations of the experiment conducted. Most notably the classification of when movement is detected is a big source of error. Given fluctuations in reported position when stationary, a coarser resolution of movement had to be used to establish movement. Although measured time showed quite long delays in some cases, the perceived behavior of the robot was not as sluggish. Final conclusion will be reserved for after subsequent tests have been conducted.

**3.2.2 Stopping Time**

A similar test with round-trip-time measurements has been conducted. In this case, the robot is in motion, and a stop command is sent. This is in essence just a special case of the first test, but differs sufficiently for it to be conducted as a separate test. Like in the stationary-to-moving tests, different values for initial velocity, movement path, and deceleration will be used.

**Initial Data**

In the documentation for the UR5, some values for certain safety features, including stopping of robot movement are described fairly detailed, and gives some expectations of what sort of
results should be anticipated. Most noticeably, an emergency stop should follow keep within the limits described by the following graph.

![Figure 24: Showing relation between time and joint speed of the UR5% after an emergency stop. Adapted from manufacturer documentation.](image)

The graph shows the joint speeds allowed in the robot after a given time following an emergency stop. When an emergency stop is triggered, there is a reaction time of 24 milliseconds before deceleration of the motors is started. Following the start of the deceleration, the system is allowed 500 milliseconds before all joints must be stationary. These values are in the case of the robot moving at maximum joint speed in normal mode.

In the UR5 user manual, the scenario of stopping the robot from full speed, with full extension of the joints, and with a 5-kilogram payload is described.

<table>
<thead>
<tr>
<th>Joint</th>
<th>Stopping Distance (rad)</th>
<th>Stopping time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint 0 (BASE)</td>
<td>0.31</td>
<td>244</td>
</tr>
<tr>
<td>Joint 1 (SHOULDER)</td>
<td>0.70</td>
<td>530</td>
</tr>
<tr>
<td>Joint 2 (ELBOW)</td>
<td>0.22</td>
<td>164</td>
</tr>
</tbody>
</table>

![Figure 25: Showing stopping distance and time of three of the UR5 joints from maximum speed with 5kg payload. Adapted from manufacturer documentation.](image)
The test of joint 0 was conducted with a horizontal movement, while the tests of joint 1 and 2 were done with a downward vertical movement. Given the design and implementation of the UR5, theses movements will be the worst case scenarios for the respective joints.

Results of Stopping Time

The test for delay and reaction time for a stop command were conducted with different velocity and deceleration values, and following different movement paths. Complete data can be seen in the appendix. In addition, the stop commands were issued at different points on the movement paths do mimic more realistic conditions. The experiment gave the following results.

![Diagram showing distribution of results of stopping test](image)

The first graph shows the distribution of stopping time when the robot was moving at moderate velocity. Values chosen for moderate velocity are above what is deemed appropriate for normal operations with users, but still speeds within what could be used in other operations. Results show that most stopping times keep within the predicted 524 millisecond limit from the specification of the robot.
When increasing the velocity of the robot to higher values, far beyond what would be responsible to use in operations directly with humans, results are quite different. Firstly, the lowest measured values for stopping time are substantially higher than in tests with lower velocity. The spread also seems bigger. It is of note that the robot is mounted on a table with wheels, and is not completely immovable which was observed in the case of high velocity movements.

The values for maximum and minimum time elapsed between stop-command issued, and complete stop detected:

\[
\begin{align*}
\text{Maximum time: } & 1778 \text{ milliseconds (649 ms at moderate velocity)} \\
\text{Minimum time: } & 210 \text{ milliseconds (700 ms at high velocity)}
\end{align*}
\]

The average of the entire experiment, including different values for velocity and acceleration is:

\[
\text{Average time: 562 milliseconds (339 ms at moderate velocity)}
\]

Other general results of interest at moderate velocity:

\[
\begin{align*}
\text{Median: } & 328 \text{ milliseconds} \\
\text{Standard deviation: } & 74 \text{ milliseconds}
\end{align*}
\]
Analysis of Stopping Tests

Theses test are as mentioned fairly similar to the stationary-to-moving tests, only in reverse. This would suggest that the same sources of error would be present in this case. Firstly, the same rounding is done to stabilize position, which in these tests can have the opposite effect. If the deviation in measured position of the robot is less than the set rounding threshold, a complete stop is registered. Ultimately, this results in a logged stop time potentially lower than the actual time it takes for a complete stop. The same potential timing errors are present here as in the other tests.

In the cases of high velocity stops, these tests showed that the base which the robot is mounted on is not completely stable. In some cases, the entire table moved several centimeters when the robot applied the brakes. This could potentially alter the results gather from the tests, and might therefore not represent the stopping time in ideal conditions.

Analyzing the results from the moderate velocity tests show quite low variance in stopping time. The experiment is not extensive enough to draw final conclusions from the results, but does seem to indicate a fairly consistent stopping time given similar circumstances.

As in the other tests, absolute conclusions for the robot in ideal conditions are hard to draw due to all potential sources of errors. However, the results do represent the environment and parameters the system has to function under, and in that sense do provide valuable information.

Conclusion on Stopping Time

Given the results from testing stopping time from moderate velocity, the product documentation of the UR5 seems to be accurate. The stopping time is also deemed to be satisfactory for the use intended for the system. High velocity testing was done to establish the impact increased momentum had on stopping time. The values for velocity used in testing will never be used in normal operation in this project, as it is deemed clearly impractical and even dangerous from observations of robot behavior.
3.2.3 Reaction Time Consecutive Commands

The last test in the evaluation of delay and latency covers the reaction time of the robot and handling of consecutive commands. Firstly, it is determined how flooding the server with commands is handled. This is done by sending commands with no time delay to separate them.

When the consecutive command handling is conducted, the time necessary between commands needed for the robot to react to both commands is determined. This is done by sending a movement command quickly followed by a stop command. Doing this with different delays between the two commands will show which time interval is necessary for both commands to be handled.

For these two experiments the time interval will be set and commands will be handled from the controlling PC. Results of the first test will be based on observing the behavior of the robot given a flood of commands. The second experiment is after initial testing decided to be evaluated by when the robot can be heard and felt moving. A characteristic sound and nudge are heard and felt whenever the robot activates motors in the joints. This approach to determining reaction was deemed superior to using output from the real-time server due to fluctuations in reported readings and the subsequent need for rounding off data.

Results Consecutive Commands and Reaction Time

Results from sending consecutive commands showed that the latest command will always overwrite the one before it. Sending two commands straight after each other results in the robot behaving as if it just received the last one. The same behavior is seen with several subsequent commands being sent back-to-back.

As long as new commands are being sent to the server, commands will not be executed. It was not tested if a system crash could be provoked using the method of flooding commands.

After the consecutive command behavior was established, the testing for finding the shortest interval between executed commands was conducted. The interval started at 200 milliseconds, and followed a binary search pattern (100ms, 50ms, 75ms, 62.5ms) until no motor activation was observed. At this point, manual increments were made to determine at what point
activation could be observed. Throughout five tests the results did not vary. Every time activation was observed at 63 milliseconds but not at 62 milliseconds.

**Analysis of Reaction Time**

Since these tests are based on physical observation from the authors, it is of note that there is some doubt in relation to the quality of the results. To comment on this, both authors agreed on all observations, and the difference between activation and no activation of the motors is fairly noticeable. Buzzing appears from the joints and a clear nudge can be felt.

The results from testing reaction time and consecutive command behavior give good indications for what is possible in controlling the UR5 over TCP/IP. First they show that, within reason, flooding of the server is not a problem. Flooding is seemingly ignored, and the last command received is carried out as usual.

Furthermore, the results showing a reaction time of 63 milliseconds gives legitimacy to sending frequent updated commands to the robot. 63ms could suggest an internal updating frequency of 16Hz. These results give an indication on how far apart control commands can and should be sent.

**Conclusion on Reaction Time**

After testing stationary -to-moving behavior of the system in earlier tests, a concern of perceived latency of the robot appeared. In the reaction time experiment however, most of these concerns were laid to rest. The results show the approach of consecutive commands does work as foreseen. Results from this test will directly impact the rate of which updated commands will be sent to the robot.

**3.3 Testing of the Force Sensor**

The UR5 robot used for this project comes with internal force sensors. These have by previous project at the department been deemed to not be precise enough to rely on for accurate measures. [19]

Another force sensor has been acquired, and is attached as an add-on. The ROBOTIQ FT 150 force sensor added is described in detail in chapter 2.3. The specifications given from the
product documentation is deemed to be adequate for use further use in the project. Any extensive testing to validate product documentation is not conducted. In later testing the force sensor is used in combination with position data to establish servo viability.

### 3.4 Proof of Concept

A considerable part of this project is based on the assumption that the robot chosen is possible to control in the desired fashion. This firstly includes the ability to perform seemingly soft and fluent movements. This extends into ideally being able to update the course of the movement without the movement stopping or stuttering to any noticeable extent. This is a metric which is not easy to measure, so this will be judge subjectively by the authors.

Furthermore, the robot will have to have an acceptably low delay on control instructions, and on outputting data to be used in control calculations. To establish this, both measurements found in testing of the platform, and a subjective opinion based on observations will be used as a basis.

Another central part of the intended solution, is the use of force data collected from one or both force sensors. This data is crucial for giving meaningful evaluation of some of the operations on the robot. The force data is also desired for use with the planned implementation of the servo-function. This will be discussed further in chapter 3.4.1.

These are some central point which if proven to be possible, will validate the concept behind this project. Since most of the criteria, although some can be measured accurately and reliably, have no set scientific threshold, the subjective observations and decisions will be described and defended.

**Conducting Proof of Concept Testing**

It is already known that the UR5 is capable of smooth movements when given a full scripted program. What needed to be proven was that the same behavior is present when controlling via commands over TCP/IP. Especially when dynamically updating movements while the robot is in motion.

A fairly simple test was conducted to establish this. The robot was given a command to move in a certain direction, and while moving, another command was sent. This was done with new
commands varying in direction to see different scenarios. Both linear movement and circular movement was tested.

Going further, the experienced delay of command the robot was noted. While conducting tests for commanding the robot over TCP/IP, delay from sending commands until change was observed was paid attention to. In addition, a small program was developed to check the update frequency of the output data from the robot.

**Results for Proof of Concept Testing**

It was quickly established that the UR5 was still capable of performing smooth movements when controlled over TCP/IP. As long as the change in direction is relatively small, turning is fairly soft. Given a larger change in direction, the robot will stutter.

The experienced delay on sent commands was low. When updating paths or stopping the robot, changes seemed almost instantaneous.

Results from the output server monitor program showed an update interval of 8 milliseconds repeatedly, consistent with the 125Hz frequency stated in the product documentation.

**Conclusion on Proof of Concept**

The results from the proof of concept testing gave no indications of needing to change the fundamental principles of the system set in pre-development. Behavior of the robot was satisfactory smooth and responsive to new commands. Output information was received at the advertised 125Hz which more than meets requirements for the project.

**3.4.1 Additional Test for Servo Viability**

After the concept is proven, additional validation of the viability of the servo approach will be done. The potential desynchronization of the data from the two sources, in combination with latency throughout the system will have an impact on how responsive the servo function can be. To test this, a test logging force, speed and time will be conducted.

In detail, whenever new data is received from the robot, TCP speed from the robot, force from the sensor, and a timestamp will be logged. This will be done during a servo induced movement and results will be stored for analysis.
This testing will offer more detailed results for the viability of the servo functions, and will potentially slightly alter the implementation.

**Results from Servo Viability Test**

The results from testing show the correlation between force applied to the sensor in the direction of movement and the actual TCP speed of the robot. The test was done over runs of 10 seconds, and with linear movement paths of the robot.

![Graph showing correlation between force in movement direction and TCP speed, system in servo-mode.](Figure 28)

**Analysis of Servo Viability**

Testing the concept of controlling movement of the robot from force input showed it possible. However, it was desirable to further establish the practical viability of this in regards to responsiveness and delay. The results from the test are visualized in Figure 28.

From the graph it is clear to see the correlation between the force in movement direction, and the actual TCP speed in the movement direction. Over the ten second span of the test, the robot is accelerated and decelerated several times by applying more or less force to the sensor. Releasing the handle, resulting in close to no force applied, the robot speed will drop to close to zero, as illustrated by the graph.
In the experiment shown in graph, the movement was finished at the last point where TCP speed hits zero. Any force applied after that point does not result in any movement. The results show some delay between force input and TCP speed, but this is believed to be mostly from the acceleration of the robot. The acceleration values are set at a level which gives fairly smooth movement and tries to eliminate jerky, uneven direction changes.

**Conclusion of Servo Viability**

The results from using the servo function based on force measured to control the speed of the robot gave satisfactory results. Some delay can be seen in the results. It is however thought that most of the delay is due to acceleration of the robot. We could not see any indications from the results which would warrant a change in approach to the servo function. The solution is concluded to be viable for further development.
4 Implementation

This chapter will present the implementation of the solution developed throughout this project. First the premise and challenges with the robotic platform will be discussed. Further, a description of definitions made for concepts used in the implementation follows. The rest of the chapter will show choices made and the implementation of both logic and user interface.

4.1 Challenges with the UR5

The Universal Robots UR5 has harvested a lot of praise in the community for being cheap, easy to implement and safe. During work on this project, and by reading other work done with the UR5, it is however clear this robot is not without flaws. In this chapter we will discuss the weaknesses and shortcomings of the UR5 as we found them, and some noted by others.

UR5 Software

Universal Robots do supply updates and patches fairly regularly, and since we started working with the robot about a year ago, several software updates have been made available. However, in our experience, the system is not as stable as we would have hoped for. Throughout or entire development phase, and especially in testing, the system was prone to different types of system crashes. Some of which the UR5 was able to recover from, but most would require a full reboot of the entire system, entailing several minutes of waiting.

The aforementioned crashes would for the most part play out by our control application not being able to establish a full connection, or sometimes no connection, to the servers in the control box. This was followed by the attached tablet control panel partially, or totally, freezing up, and require a restart. Sometimes the control tablet alone would freeze, and sometimes just the servers. These errors could possibly be provoked by some bad handling of connections or other faulty code in our own developed software, although in testing against a server created for testing purposes, we could not observe any severe problems in any of our test scenarios. This leaves us to think that at the very least some insufficient error handling is present in the UR5- or control box-software.
**UR5 Movement**

We would further argue that the internal movement controller, and possibly kinematics engine has some shortcomings. During testing, or even just moving the robot around with the on-board controls, we encountered frequent emergency stops. These could be due to excessive joint speeds, trying to move out of range, or encountering singularities the robot could not handle.

In addition, the UR5 is not sufficiently *self-aware*, by which we mean it does not know where itself is. This has been very apparent in testing, with frequent collision between different parts of the robot arm. Although these problems have been made worse by the mounting of the force sensor and other various equipment on the arm, the problem is still very apparent without any additional equipment mounted.

Further problems observed with the movement of the UR5 was with the tool speed exceeding the safety limits without the command sent explicitly informing it to. The cause of this problem is not completely known by us, but it is thought to be a problem somewhere in the kinematic conversion.

### 4.1.1 Safety Concerns

As stated in chapter 2.2, the UR5 has a lot of safety features. These include settings for maximum speeds, acceleration, the ability to define allowed area of movement, and a maximum force setting. In the setting of this project, where the robot is meant to be working directly with humans, these safety features are indeed appreciated.

However, these have not shown to be completely covering for all scenarios possibly resulting in physical injury or harm. The most apparent problem, also acknowledged by Universal Robots, is that none of the safety features takes momentum into account. Moving at a fairly high speed and with a payload of several kilograms, the momentum could be considerable. The question, “Should we fence the arms of Universal Robots?” is posed by Ilian Bonev at Control and Robotics Laboratory [49]. By using an optical tracker and force measuring equipment, they found that the force of impact with a 5-kilogram payload, and speeds of 0.5 m/s, could be as high as 1500 N. They note that the impact was metal to metal, and the object hit was fixed, but that the force experienced by a person being hit in the same scenario, could indeed be dangerous. Furthermore, there is no internal check in the robot to verify that the
weight of the payload corresponds to the weight input. This can result in the robot smashing in the ground when joint brakes are released if the weight is heavier than entered, or the arm swinging upwards if the weight is lower than expected.

Another hazard we have observed is getting caught in between the moving parts of the robot. If this happens close to the joints, the forces trapped body parts could be subjected to could be enough to harm the person.

4.2 Core Operational Elements

This chapter will establish some of the core concepts and elements as used by the authors in the thesis and in operation of the system developed. This includes the explanations for what these are and how they relate to each other.

Commands

A command is in essence one single line sent to the robot from the control application. Such a command contains information on what the robot should do next. This could be a movement to a certain point, going to a complete stop, or setting certain internal variables on the robot. Commands are used to issue each new step in an exercise, or in initial moving or setting up of the robot.

Exercises

An exercise is defined as movement between at least two points in space. Each exercise will have a set purpose in the rehabilitation or training of a user. Exercises are made up by a list of points in space as well as information about stops in the movement. The exercises can be created directly on the system or transferred from another system. Each exercise will be loaded into the application and executed through logic implemented within the application.

Servo Function

In addition to the functionality of running exercises on the system, an implementation of a servo function will be done. Such a servo function will have two main applications as we see it. The first will be adding a servo to an exercise. By this we mean the robot should aid in movements, but also dynamically alter the movement based on the interaction of the user. By
adding this functionality, the robot will not simply follow a set pattern of points in space at a set speed, but adapt to the physical input of the user.

Another similar implementation of a servo function is a free-form servo function. This would put the robot in a mode where it moves solely based on physical user input, and not based on a list of movements like in an exercise.

The approach to implementing such servo function is using the force sensor data to establish what force the robot is subject to at any point in time, and deduce the direction and force of physical input from a user. Based on this processed data new commands are sent to the robot containing new information about movement direction or speed.

### 4.2.1 Data Storage

This chapter will focus on data storage in the system. Alternative solutions for implementation of storage will be discussed, as well as potential challenges with this aspect.

As in any development project, there will is a need to store data. Storage needs to meet the criteria for the given system in regards to accessibility, efficiency and flexibility. In our case specifically, there is also the added concern of potentially handling sensitive information about the users, patients, of the system.

**Handling of Sensitive Patient Information**

Since the system developed is meant for interaction with and handling of patients, there is an immediate concern with storing this information. In theory, substantial amounts of information about the patients could be found in the system. At the current prototype stage there is however no data tying to any individual in the application. Secure storing of such information is still a factor in choices made for the implementation.

**Databases**

The system developed in the current iteration does not make use of databases for storage. In a later iteration introducing user profiles and storage of more information databases would probably be introduced. At the current stage of the application it was deemed unnecessary to implement databases as it would not add any obvious benefits to the system as is.
Serialization of Objects

In the prototyped application, the need for storage is limited to serialization of object for use in later instances of the application. Primary this is in the case of saving exercises for later execution. It was decided that a serialization to the XML-format would be suitable for this purpose. XML is accessible for viewing for operators and administrators, and can incorporate several security solutions if needed at a later stage.

4.3 Chosen Control Method

As stated previously, there are different ways of controlling the UR5-robot. Including URScripts running on the robot, sending commands over TCP/IP, and using a custom robot controller with a C-API and loading them onto the robot. To make a well informed choice of which control method to go with, the different alternatives were evaluated up against some central criteria for the solution. The criteria are not ordered by priority.

- Responsiveness
  The control method chosen would have to be responsive enough as to deliver control adequate for interaction with humans. The system should not be experienced as slow or jerky.

- User Experience
  Given the user base of such a system, user experience and accessibility are important for the system to be used to the degree it is meant.

- Flexibility
  The controls need to be flexible enough to cover all likely user scenarios without extensive modifications needed. It should also be possible to dynamically alter the behavior of the robot with relative ease.

- Extensibility
  It is crucial that the control scheme allows for easy extension of the solution by adding new software modules or additional hardware.

- Safety
  A system aimed at working in direct contact with humans need to be satisfactory safe. Although the UR5 itself includes several safety features, safety should be considered in all layers of the system.

Although the criteria have not been prioritized in any specific order, some of them could potentially result in ultimatums trumping other criteria. An example is if safety is
compromised by going down one path for controlling the robot. Even if this path would give better potential on all other criteria, the option would probably fall through. The final decision will be made from evaluating all the central aspects together, and the option with the biggest potential, and no fatal flaws, will be chosen.

4.3.1 Controlling Through the Supplied Control Panel

One of the central aspect of this project was to make the solution user friendly, and as familiar as possible to the user. The tablet included with the robot, and the onboard graphical user interface is both simplistic and informative. Unfortunately for use in this case, it is deemed to technical for non-technical user operation. Controlling the GUI from the inductive touch screen does suffer from not being as responsive as it ideally should be.

Controlling the robot from scripts running internally does offer several advantages. First of all, the scripting language is developed with that specific task in mind. This presents us as developers with a wide range of methods and controls out of the box. As an example, moving the robot through a set of waypoints at different speeds with varying pauses is as simple as writing the list of waypoints with a few parameters. However, one central ability of the system, being able to dynamically alter or create behavioral patterns of the robot, was seen as being too impractical with this approach to control.

Additionally, since the solution should be expandable with software modules such as the servo functionality and gamification principles, using the attached control tablet and running scripts on the robot from there, is not eligible. Although the scripting language does include support for fairly advanced logic, and integration with the force sensor if relatively easy, this approach was seen to have too many limitations as far as extensibility was concerned. Furthermore, there is no way of knowing if additional hardware is possible to integrate with the hardware on the robot.

As far as safety is concerned, using the on-board controls does not compromise any of the included safety features, and it is possible to implement custom safety methods in the programming. In addition, the control panel is mounted with an emergency stop button.

The option of controlling the robot exclusively from the included software and control panel does is concluded to give easy implementation of a lot of the functionality needed for the
system, and does uphold the standard of safety needed. Unfortunately, this approach is also limiting in flexibility and extension of the system.

### 4.3.2 URScript

As described in the previous chapter, the UR5 can be controlled through URScripts running on the on-board controller. These scripts can be created or modified in a separate remote application, and uploaded to the robot for execution. Using this control method is fairly similar to using the control panel in other aspects as well. The areas where it differs will be described further.

This approach does offer most of the advantages of controlling through the control panel, but with some additional possibilities. Data from the robot can be read to a computer and dynamic calculations can be made. Since all logic except the direct controlling of the robot can be done in a separate program, this approach is quite flexible.

In terms of user experience, this control method has very few limitations. Any design decision made in a control application are separate from the controller. This offers possibilities for extending functionality later.

### 4.3.3 C-API Controller

Controlling the robot through the C-API provides the fastest response out of all the different options. This approach gives the developer full freedom in almost all regards, including controlling motors in the joints, writing custom kinematics engines and more. Although this does give almost unlimited potential as far as functionality is concerned, it does also give rise to a lot of challenges.

There is an implementation of the C-API available through the ROS environment. However, we found there to be too little documentation, too much overhead, and generally too many disadvantages in our case.

Later in the development phase, it was disclosed by Universal Robots that the C-API support would be discontinued. It was expressed a wish of moving all development over to the scripting API, and improvements to responsiveness and overall performance of the scripting API would be implemented.
4.3.4 Controlling Over TCP/IP

The option of controlling the UR5 with commands sent over TCP/IP was seen as the one giving the highest ratio of degrees of freedom to complexity and cost in development. Going with this control schema opens up an easy way to control the robot while being able to abstract most of the core logic to an external computer.

By having the user control everything from a PC, the user experience will be closer to prior experiences the user has had with computers. Additional and separate GUIs could be developed for different user groups, and expansion modules could easily be added later.

The main concern with this alternative, is responsiveness. In principle, controlling the robot in such a way is a remote control approach, since the commands between application and robot are sent over TCP/IP. Certain problems could arise with this. Both delay of communication, and loss of connection can pose a risk to the stability of the system. Since the communication is done over what can only be judged as a tried and tested protocol, given a short cable span of a few meters and only one router in the communication link, it has been evaluated that this will not be a significantly weak link in the system. This is discussed in further detail in the chapter on evaluation of the robotic platform, but in short, the responsiveness was found to be satisfactory for the task.

Safety, however, could potentially be compromised by a loss of connection between the controlling application and the robot itself. This concern is mitigated first by the presumed low risk of break in communication, and secondly by the fact that all internal safety features of the UR5 is upheld even if such a break should be experienced. Only additional safety introduced by the application logic will be temporarily lost.

Since the logic of the solution is situated outside the robot and control box, there is a need for getting information from the robot regarding positions, speeds and states. This is possible through the servers running in the control box of the robot.

Looking at the potential for extending the system, this option does present very few limitations. Through abstracting all logic from the robot itself, and the only link being through standardized TCP/IP communication, there is no apparent constraint on potential additions in the future.
4.3.5 Conclusion of Control Method

The different control methods deemed relevant have been discussed. After an evaluation of their respective pros and cons, the decision fell on controlling the robot by sending commands over TCP/IP. This control method was chosen mainly due to the possibilities for extensibility and the overall flexibility of the approach.

Since the controlling application will run on a PC, the design and development options are countless. In addition, we could not see any clear disadvantages in moving most logic out of the robot.

4.4 Communication Between the Sub-Systems

The entirety of the system consists of three distinct and, in part, separate systems. These are the robot with control box, the PC with control application, and the force sensor. All are discussed further in chapter 4.7. A central aspect to the system as a whole is therefore the seamless integration and incorporation of the separate physical and logical parts into one system. To accomplish this, communication between the different entities is key.

4.4.1 Communication with the Force Sensor

The ROBOTIQ FT-150 force sensor as further explained in chapter 2.3, is a separate, stand-alone part mounted to the robot. Given power, the sensor works regardless of the states of any other entity in the system. Data to and from the FT-150 is sent over the RS485 standard, and converted via an adapter to USB for easier integration with other units.

On the receiving end, in our case the control application on the PC, the USB signal is handled as a standard serial communication connection using built in .NET libraries. MODBUS commands are then used to put the sensor in a ‘stream mode’, initializing the data stream of force reads at 100 Hz.

4.4.2 Communication with the Robot Control Box

All communication between the controlling PC and the robot is done over TCP/IP. Both the robot control box and the PC are connected via Ethernet cables to a designated router.
mounted to the robot stand. All connected nodes have static IP-addresses for predictable communication.

As mentioned in chapter 2.2, there are different servers, with different responsibilities, running in the control box. The control application running on the PC keeps track of the different connections to the different ports and servers. All connections are done one asynchronous sockets for a more flexible result.

4.4.3 Challenges with Data Synchronization

When working with several separate sub-systems and incorporating them into one, there are a few concerns that need to be taken into account. Perhaps the biggest one is the challenge of temporal synchronization of data. In this system there is two-way communication between entities, and no central shared clock.

The control application works with two major data sources, the force sensor, and the ‘real-time’ server in the control box of the robot. Both data sources are running in modes similar or resembling to data streams. The robot server sends data packets over TCP/IP at a rate of 125 Hz, while the force sensor sends data over serial communication via USB at a rate of 100 Hz.

Perhaps the most apparent notion here is the fact that the different data streams will rarely sync perfectly given their different refresh rates. However, the differences in stream frequencies are small, and given the absolute rates, the deviation attributed to this were deemed small enough to ignore in the current setting. Further, the actual transportation of the data could be a contributing factor to desynchronization of the data. This problem is potentially even worse due to the different protocols and even physical layer of data transportation between the two sources. A third potential problem could be if the processing of the received data was of different complexity for the different sources. This is however not the case in this setting where data from both sources is fairly straight forward to process and does not need any time consuming computing.

Taking all these factors for temporal desynchronization of the data streams into consideration we still found a fairly simple approach to be satisfactory. Both data streams are handled asynchronously in the control application. Whenever new data is received, this data is processed in the context of the last data received from the other source. This is not guarantee
for the data to be perfectly synchronized, but we found it a better and more efficient approach than trying to match data packets in any other way.

Data going from the control application to the control box of the robot is not subject to any synchronization from our side. The commands are sent at the completion of the necessary data processing, and commands are conducted and overwritten in chronological order by the robot software.

4.5 Positioning and Moving the Robot

In addition to the method chosen for administering commands and controlling the UR5, there are also an array of different approaches to the positioning and moving of the robot. The preinstalled controller onboard the robot is able to process and execute several different types of commands which further leads to different means to, at least, comparatively similar goals. This chapter will present and discuss these differences and tie them into our solution.

4.5.1 The Robot in Space

As described in chapter 2.2, the UR5 is mounted on a table. The onboard controller has the ability to relate and interact with the world around the robot in a few different ways, including joint positions and TCP position in both fixed and relative coordinate systems. Further the ability to limit the world as seen from the robot to subsets of the whole coordinate system like two dimensional planes, spheres or even lines, can give greater control.

The possibilities of limiting the world of the robot was looked at early in the project. If an exercise could be executed in a two dimensional plane, all logic could relate to velocity and direction in a two dimensional setting, and would therefore be simplified by thereafter. The disadvantages that became apparent with this were first of all that this approach would limit the exercise to one plane. This obviously limits the degrees of freedom, and although it might be sufficient in a lot of cases, there would be a need for handling of the cases where it would not be. Such a solution would lead to two different program logics, one for general cases and one for special cases. It was deemed unnecessary to implement this, since the logic for handling the special cases, should in all cases also be able to handle the general cases.
When the choice of relating the robot to the full coordinate system in all cases was made, we looked at different possibilities for the mapping of the space. It was early decided to fix the main Cartesian coordinate system to the base of the robot as this seemed the most intuitive. To keep the model as intuitive as possible, we further chose to use absolute coordinates as points in space, and not relative to current position. This would make the avoidance of known ‘problem areas’ easier as well.

4.5.2 Movement Commands

Since the robot is being controlled over TCP/IP the movement commands available are limited to those possible through the URScript language. The ones chose for evaluation in the project are ‘movej’, ‘movel’, ‘movep’, and ‘movec’.

The movej command gives the robot control the ability to freely choose the way the joints move to get the TCP in the desired position, and is therefore believed to be the best way to avoid singularities and self-collision for the robot. However, this way of movement does not give any guarantee for which path the tool of the robot will take to the desired destination, and is not suitable for movement in exercises.

Both the movel and movep commands differ from movej in that the robot controller will calculate the movement of the joints to enable a linear path to the destination for the tool. This makes these two movement command usable for linear movements in exercises, and full control of the tool path. Unfortunately, this approach seems to, not proven, result in more self-collisions and singularity problems while moving. Using movep instead of movel gives the robot the ability to not stop before going on at the goal waypoint, resulting in smoother chaining of movements.

Given the nature of physical exercises, linear movement does not completely cover all the necessary movements. This is why the movec command is included in the project. One possibility to mimic round or arched movement paths is by chaining many short linear movements. However, the ability to use the built-in movec command, giving a circular movement via a given middle-point, could result in smoother movements, and does reduce complexity in control logic.
Through evaluation of the different movement commands sufficient variations between them gives rises to scenarios where all three major types, movej, movep and movec will be useful. By adding support in the control application for all three we believe additional value will be added, and that the complexity cost of including this will not be considerable.

### 4.5.3 Approach to Exercises with the UR5

One of the main use cases of the robot assisted rehabilitation solution is running physical exercises. There could be several approaches to both creating and running these exercises. The first alternative looked at was adding a set of exercises to the program. These exercises would be based on a certain movement trajectory. Depending on input criteria like arm length or sitting position of the user, the system would dynamically calculate waypoints. This was somewhat the approach taken in the bachelor’s thesis this project is based on [19], and the concept has been tried and proven there. However, we deemed such an approach to be too limited in adding new sets of exercises. Furthermore, we did not feel confident enough in that such a purely mathematical approach to the exercises necessarily would give optimal results in practice.

Another approach, perhaps more intuitive as well, was seen as a better option. By simply constructing exercises by adding points in space, the user would be more likely to get the result anticipated. This is done by using the servo-mode developed allowing the robot to move freely in any direction. The exercises would then be stored individually for each user or group.

The drawback of this approach is the added complexity in setting up the system for a user by having to construct exercises each time. To counteract this the process of creating exercises should be as streamlined and functional as possible.

### 4.6 Programming Approach

This chapter will present some central aspect of the programming approach in this project. A short description and background for the choices will be given.
Tools

Both authors have some previous experience using the C# programming language. The impression is that it is an easy to use programming language that is part of a well-supported platform. As part of the .NET framework, it comes with an extensive native library, and there is a vast amount of libraries and APIs available from third party groups and suppliers. Given the authors knowledge and experience about this and other programming languages it was decided that using C# would be the best option for this project.

IDE

Visual Studio is Microsoft’s own IDE. Microsoft being the developer behind the .NET programming framework and the C# programming language, it is our impression that Visual Studio is the IDE that is most suited for the task. The IDE offers built-in compiling and extensive debugging tools.

Source Control

Being a joint effort, the use of a source control greatly simplifies the process of teamwork on the project. The main purpose of the source control is to have a shared space for the project that allows for cooperative work, version control and handling of conflicts. It was decided to use Team Foundation as a source control because it is an integrated part of Visual Studio and delivers the required functionality.

Software Testing

Building a piece of software requires constant testing of the functionality. Much of the software in this project requires testing on the UR5 robot in order to verify that it works as intended. This includes verifying that the commands are properly constructed and that the robot moves in the expected manner. Some of the logic requires data from the robot, but does not require the robot performing any functionality in order to verify its function. Due to not having the robot available at all times, certain steps have been made to make sure functionality can still be tested.
**Test Server**

One of the steps that has been made to ensure that functionality can be tested even without the robot, is creating a test server. The purpose of the test server is to approximate the data that the robot is distributing, such that it facilitates the purpose of the functionality being tested. Different types of dummy data have been used corresponding to the different types of functionality being tested.

In addition to supplying dummy data, the test server has also been useful as a way to test and verify the logic of the asynchronous client. When testing on the robot, flaws in the logic of the client was not immediately obvious. Having our own server to test on provided the ability to see exactly what was going on server side, and how the server reacted to events in the program, like disconnects, both controlled and due to exceptions.

### 4.7 Overview of the System

This chapter will present the different top level of the system designed and developed in this project. We divide the solution into three main parts; the robot with control box, the force sensor, and the PC running the developed application.

![Diagram showing simple overview of the system components.](image)

*Figure 29 Showing simple overview of the system components.*
In early development stages of the project, the decision was made to keep design and implementation of the solution at a modular and abstracted level. Keeping with the initial plan of controlling the hardware, in this case the UR5 robot, from an external computer, we saw the added advantage of the application being adapted to different hardware with changes mainly in the communication modules. This approach is also in line with our understanding of good practice in software development.

**ROBOTIQ F150 Force Sensor**

Discussed more in depth in the hardware specific chapter 2.3, we will not elaborate too much on the F150 force sensor here. The sensor is connected towards the end of the robot arm, just before the mounted tool. Communication with the sensor, starting the data stream and reading data, is done over a serial interface converted to a USB-connection. This connection is directly between the controlling PC and the force sensor, while the only connection between the sensor and the robot, is the power supply given through one of the digital outputs in the UR5-control box.

**UR5-Robot**

A more detailed description of the UR5-robot can be found in chapter 2.2. The UR5 is not directly connected to the controlling PC, it is controlled through the control box. It is physically mounted on a table, and both the control box and force sensor are part of the same physical installation.

**UR5-Control Box**

When using the term control box, we mean everything inside the box the robot itself is mounted to. This includes servers, robot controller, I/O panels, as well as the power supply. The control box is the main interface for controlling the robot, and is also responsible for pushing information about robot state, speed, position and so on, to the control application.

**Control Application**

The control application is at the center of the project. It is designed to be run on any PC running Windows, and to be the only software needed to turn the UR5-robot into a system for
robot assisted physical therapy. Installation of the entire system should be as simple as installing the application, and connecting the two cables to the PC.

4.8 Introduction to System Components

This chapter will give a short introduction to the different system components of the control application and how the work together. A more in-depth and technical description of the implementation will be presented in the following chapters.

Data Acquisition

As stated earlier, the system uses two primary data sources for all computations and processing. These two are the force sensor and the ‘real time’ server in the robot control box. The main classes in the program relating to data acquisition form these two sources will be explained here.

ForceDataRead Class

The communication with the force sensor is implemented in a ForceDataRead class in the program. This class is responsible for establishing a serial connection with the force sensor, and is based mostly on the .NET library SerialPort included in System.IO.Ports. Through the ForceDataRead class, an event is triggered every time new data is received from the sensor, and the new data is made available to the other classes in need of it through a public, static property.

In addition to the reading of the data from the sensor, the data is zeroed in this module. Since the sensor itself does not include functionality for zeroing the output to a set offset, this is handled in the ForceDataRead class. The actual data is zeroed by simply storing the momentary force at the desired point in time, and subsequently subtracting that force from all new data read.

Robot Class

To continuously keep track of the states and positions of the robot, the Robot class is in charge of the communication with the robot control box over TCP/IP. Two different TCP/IP clients are created, one for receiving data and one for sending commands the robot.
Sending commands to the robot is handled through a static SendCommand method which takes a string input argument. The command is then sent through the corresponding socket.

Whenever data is received from the robot through the connection to the ‘real time’ server, an event is raised and handled. Each received packet contains 1044 bytes, and is arranged in a specific order. After receiving the data, the necessary bytes are handled and converted into usable data. Specifically, position data from the robot is stored in a separate object for easier access from other program logic.

**Running the Robot in the Physical World**

Perhaps the most protruding part of the system is the actual running of the robot. This is handled a few different ways in the control application depending on the state of the robot, and on the objective. A few different utility classes are developed for an organized representation and interaction with the physical world.

**Points**

One of the most basic objects in the entire system are points. These are simple representations of a point in space consisting of six double-values; X, Y, Z for Cartesian coordinate position, and RX, RY, RZ for rotation of the TCP of the robot. Points are used whenever the robot relates to the physical world, and additionally contains some utility methods for building vectors and a ToString.

![Figure 30 Showing the implementation of Point.](image-url)
The link between points and movement of the robot is the class URMove. An object of this class is the representation of one movement of the robot. It contains target position, movement type, and a middle point if the movement is curved. In addition, the velocity of the movement and a direction vector is stored here.

The toString of a URMove object returns a string formatted as a command. This is sent directly to the robot to execute a movement.

**Implementation of Exercises**

The principle of what an exercise is, is explained in chapter 4.2. These exercises are stored as XML-files on the PC running the application and is loaded whenever an exercise is desired executed. An exercise object contains an array of URMove objects referred to as the route of the exercise. Corresponding to the route, an array of integers specifies if, and for how long the exercise should stop and wait at a given point in the route. Exercises also contain information about which type of movement should be conducted, and a name of the exercise.
Tracking the Robot Position

One of the challenges of implementing exercises in the system is tracking the completion of movements of the robot. There is no explicit confirmation from the robot upon reaching a destination. Following this, there is no direct way of sending a list of commands to be executed in order since the robot simply overwrites the last command upon receiving a new one in command over TCP/IP mode. This gives rise for the need of continuous tracking of the robot.

Already, as explained about data acquisition, updated information about the robot states are sent out and received at a frequency of 125 Hz. To solve the problem of tracking the robot position a tracker class is devised to constantly surveil the current position of the robot, and notify through an event when it reaches the defined destination based on an input point.

Tracking Force

In addition to tracking of the position of the robot continuously, there is also a need for keeping track of the force applied to the robot at any time.

The tracking of force is limited by the refresh rate of 100 Hz from the force sensor, and by properties of the serial communication drivers in windows. However, the actual refresh rate is
evaluated to be well within the critical range, since the command rate of the robot over TCP/IP is about 16 Hz.

**Executing Standard Exercise**

When an exercise is loaded into the application, the initial point is prepared to be tracked, and the command to move is sent to the robot. On approaching this point, the next point is loaded.

Consecutive points are handled in a run-loop following this logic:

1. Send Command: A command with the next point is sent to the robot.
2. Update Tracker: The next waypoint of the movement is sent to the tracker.
3. Tracker Notification: Program is idle until notified on the robot closing in on the destination.
4. Wait: A timer is set with the corresponding time to wait at specified point in route.
5. Timer Notification: Program idle until timer notifies to resume.

Upon reaching the last point in the exercise route, the execution notifies about completion and exits.

**Executing Servo Exercise**

If the option of adding servo functionality to the execution of an exercise is selected, a slightly different logic is running. The preparation and initial placement of the robot is identical to a standard execution. Further, the same basic steps in the run-loop are present.

The added servo functionality is added through including the force tracker into the loop. Throughout the execution of the exercise, the force from the force tracker in the direction of the movement tangent is computed. This force vector is the used to compute a new velocity of the current movement, and a new command with updated velocity parameter is sent to the robot.

![Figure 34 Showing the ExecuteExercise class.](image)
If the force in the direction of the movement increases, the new velocity will be higher than the current, and with decreasing force, the velocity will decrease. If force in the opposite direction is detected, the velocity is set close to zero, and the movement practically stops until force in the right direction is detected again. The entire exercise can optionally be stopped if the negative force surpasses a certain threshold or sustains for a set amount of time.

4.9 Implementation of Communication

This chapter will present the implementation of the different communication modules in the control application. There are two distinctly different communication implementations in use in the system; the client responsible for all TCP/IP communication with the robot directly, and a module responsible for communication with the force sensor.

4.9.1 Asynchronous Client

All communication with the robot goes through TCP/IP connections. When developing a user centered software highly reliant on network communication between entities, it is important to make sure the socket communication does not significantly impact the performance of the software. In particular, it is important to maintain a responsive user interface to make sure usability and the user experience is not diminished. In the perspective of a modern software application, socket communication can involve significant waiting times when connecting, sending and receiving data. If this communication is done on the same thread as other important functions, it can severely impact performance as network communication works on a scale of several milliseconds or more, while most computer functions work on a scale of a few microseconds. Asynchronous communication can be used to avoid blocking the main thread when waiting for a response from the server.

.NET supports asynchronous client server communication through its asynchronous programming model. The network connections are processed on separate threads, and callback methods are used to process the results on the main thread once they arrive. [50] ManualResetEvents are used if the main thread needs to suspend execution until certain actions are performed.
Connecting

Following is an example from the client class. A socket is created with the supplied IP. Its BeginConnect method is called with the endpoint of the remote host, an AsyncCallback containing the callback method to be executed when the asynchronous task completes, as well as an object to be sent to that method, in this case, the client socket.

```csharp
public void StartClient(string ip, int port, bool receiver)
{
    // Connect to a remote device.
    try
    {
        // Establish the remote endpoint for the socket.
        IPAddress ipAddress = IPAddress.Parse(ip);
        IPEndPoint remoteEP = new IPEndPoint(ipAddress, port);

        // Create a TCP/IP socket.
        socket = new Socket(AddressFamily.InterNetwork,
                             SocketType.Stream, ProtocolType.Tcp);
        // Connect to the remote endpoint.
        socket.BeginConnect(remoteEP,
                             new AsyncCallback(ConnectCallback), socket);
        connectDone.WaitOne(100);

        if (receiver)
            Receive();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
}
```

The BeginConnect tries to establish a connection to the remote endpoint. If the endpoint exists, the callback method will be called and if successful the connection will be established. The ManualResetEvent connectDone suspends the thread for 100 milliseconds or until it is signaled to continue.

```csharp
private void ConnectCallback(IAsyncResult ar)
{
    try
    {
        // Retrieve the socket from the state object.
        Socket client = (Socket)ar.AsyncState;

        // Complete the connection.
        client.EndConnect(ar);
        if (OnConnectedEvent != null)
            OnConnectedEvent(EventArgs.Empty);

        Console.WriteLine("Socket connected to {0}",
                        client.RemoteEndPoint.ToString());
    }
}
```
The callback method implements the AsyncCallback delegate which takes a parameter of type IAsyncResult. The AsyncState parameter of the IAsyncResult object contains the object that was passed in the AsyncCallback constructor, in this case the client socket. Calling EndConnect on the client with the asynchronous result completes the connection. In this example, a separate event is triggered to signal that the connection was successful to any subscribers. After connection is completed the connectDone ManualResetEvent is set, signaling that the main thread can continue from where it was suspended. It is important to note that the use of ManualResetEvents can depending on circumstances, defeat part of the purpose of asynchronous communication if it is used to suspend the main thread. ManualResetEvent should only be used to suspend a thread when functionality is about to be performed requiring that the asynchronous operation has finished its current task. As much as possible, functionality should be put in a callback method rather than after a ManualResetEvent.

**Receiving Data**

After connecting to the server, it is possible to start receiving data. The receive method follows the same pattern as the connect method, but also supplies a buffer to store the returning data in.

```csharp
private void Receive()
{
    try
    {
        // Create the state object.
        StateObject state = new StateObject();
        state.workSocket = client;

        // Begin receiving the data from the remote device.
        client.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
             new AsyncCallback(ReceiveCallback), state);

    }
    catch (Exception e)
    {
```
The purpose of the StateObject class is to serve as a wrapper so that both the socket and the buffer can be sent to the AsyncCallback method, in this case, ReceiveCallback. The ReceiveCallback method receives the state object in the AsyncState of the IAsyncResult parameter. Using the state object, it can access the state buffer that now contains the returned data from the server. Calling EndReceive on the client socket returns the amount of bytes read from the server. If there are any bytes read, it is stored and sent within a DataReceivedEventArgs object through the OnDataRecievedEvent to any subscribers. As long as there are bytes read there might be more coming and the BeginReceive method is called again on the client socket using the same callback method. The robot is continuously pushing data through its data port causing this method to be called in a recursive loop for as long as the connection remains intact. Had this not been done asynchronously, it would have throttled the main thread.

```csharp
StateObject state = (StateObject)ar.AsyncState;
Socket client = state.workSocket;
// Read data from the remote device.
int bytesRead = client.EndReceive(ar);
if (bytesRead > 0)
{
    // There might be more data, so store the data received so far.
    response = (Encoding.ASCII.GetString(state.buffer, 0, bytesRead));
    if (OnDataRecievedEvent != null)
    {
        OnDataRecievedEvent(new DataRecievedEventArgs(state.buffer));
    }
    if (!abortReceive)
    {
        /// get the rest of the data.
        client.BeginReceive(state.buffer, 0, StateObject.BufferSize,
                             0,
                             new AsyncCallback(ReceiveCallback), state);
    }
    else {
        receiveAborted.Set();
    }
}
```

If the abortReceive variable is set to true, BeginReceive will not be called and instead the ManualResetEvent receiveAborted is set. Doing this ensures that the client object is not closed while still waiting for the asynchronous callback of the BeginReceive method. When receiveAborted is set it signals the main thread to continue closing the client. Doing this avoids exceptions due to trying to access a closed socket in the callback method.
public void CloseClient()
{
    try
    {
        // Release the socket.
        abortReceive = true;
        receiveAborted.WaitOne();
        connectDone.Reset();
        client.Shutdown(SocketShutdown.Both);
        client.Close();
    }
    catch (Exception e)
    {
        Console.WriteLine("Error: " + e.ToString());
    }
}

4.9.2 Force Sensor Serial Communication

Getting data from the FT 150 force sensor is crucial for the implementation as planned. Retrieving the force reading was however a bigger challenge than expected. The sensor was used in the bachelor thesis [19] this master thesis is based on, and the implementation was thought to be straight forward. However, all robot control logic in the previous project was run on the robot through URScripts, and not commands sent from a PC application.

To use force data directly on the robot, a driver is supplied for installation on the UR5. After installing this, the sensor is plug-and-play. Our first thought was retrieving the data received on the robot through the same TCP/IP interface used for the other communications. This approach did however not work the way anticipated. Further attempts of establishing a designated connection for retrieving force data was made. No data was collected through these methods.

It was then decided to use the USB interface form the force sensor to connect directly to the controlling PC. Development was started on an application communicating with the force sensor over USB. The manufacturer provides a developer kit on their website, however, this was not a completely documented developer kit. After quite some time without any substantial results from the USB application, a forum discussion about the same problem was found. In this discussion the consensus seemed to be to go with MODBUS protocol over Serial Port over USB.

.NET has native support for serial communication through the SerialPort class in the System.IO.Ports namespace. It works by setting up a SerialPort object with the correct COM
port number, the baud rate, which is similar to the bitrate, the type of parity bit, and the type of stop bit used. With the SerialPort object initialized, it is possible to send and receive commands and data through writing to and reading from the port. A DataReceived event is supplied so that you don’t have to continuously check the port for data, and instead only read from it when data has arrived.

```csharp
public static void DataReceivedHandler(object sender, EventArgs e)
{
    if (port.BytesToRead >= 32)
    {
        byte[] buffer = new byte[port.BytesToRead];
        port.Read(buffer, 0, port.BytesToRead);
        int[] dataOut = new int[6];
        int startindex = 0;
        bool startFound = false;

        //find start of new data
        while (startindex <= 16 && !startFound)
        {
            if (buffer[startindex] == 0x20 && buffer[startindex + 1] == 0x4E)
                startFound = true;
            startindex++;
        }

        //retrieve data from buffer
        for (int i = 0; i < 6; i++)
        {
            dataOut[i] = BitConverter.ToInt16(buffer, startindex + 1 + (2 * i));
        }
        ForceData = dataOut;
        port.DiscardInBuffer();
    }
}
```

In order to receive data from the FT 150 force sensor, a particular byte sequence needs to be written to the port. This byte sequence puts the sensor in stream mode. The data is then received as six values representing the X, Y, Z, RX, RY and RZ measurements of the sensor.

Since the sensor has no zeroing function built in, data is offset by a set of readings from a certain point. This is done by calling a ForceToZero function, which simply sets the offset data to the current force.

```csharp
public static void ForceToZero()
{
    _offset = _data;
}
```

All subsequent reads will then be seen as zeroed from those values by the system.

84
private static int[] getNormalisedForce()
{
    int[] normalisedForce = new int[6];

    for (int i = 0; i < 6; i++)
    {
        normalisedForce[i] = _data[i] - _offset[i];
    }

    return normalisedForce;
}

RobotData Class

The RobotData class is the container class for all the data directly related to the robot. Data received from the robot is processed and stored here through its setData method. This method takes a byte array straight from the buffer of the TCP/IP connection and translates it into meaningful data like the position and speed of the TCP (tool center point).

This class is also responsible for creating the position tangent; a vector pointing in the direction of the intended, actual or previous motion of the robot. If the robot is in motion the position tangent is created by subtracting the vector of the previous registered position from the current position. If the robot is currently motionless it will either use the previous position tangent, or if that is not available, check the most recent movement command and use its associated direction vector. If no recent command is available either, there is no logical position tangent and it will be set to the zero vector.

Force Class

The force class is the container class for the force data. It provides properties for easy access to the different force components as well as a ForceInDirection method that returns the magnitude of the component of the force data that goes in the same direction as the vector supplied in the parameter.

    public Double ForceInDirection(Vector3D directionVector)
    {
        directionVector.Normalize();
        return Vector3D.DotProduct(directionVector, this.ForceVector);
    }

It is also responsible for translating the force data into the coordinate system of the robot. Since the TCP always has a constant rotational position that is aligned with the robots axis,
doing this only requires a simple inversion of the axis on the force data and a swapping of the x and y-axis.

4.10 Our implementation of MVVM

It was decided that using the MVVM design pattern would be a good idea for the project because it allows for a good separation of concerns. In particular, it is good for separating the user interface from the rest of the design. It allows for better compartmentalizing the design, making it easier to work individually on separate parts of the program. Using this design pattern should also make the addition of updated or new user interface modules easier.

To keep a clean ordered project, the different files are put in directories corresponding to their place in the MVVM architecture, Model, View and ViewModel. View being used for the user interface files, consists of XAML files and their corresponding code behind files. ViewModel contains a unique ViewModel for each View and a ViewModelBase class that all the ViewModel inherits from. Model contains most of the business logic and all the core functionalities of the program.

Figure 35 Showing relations between Viewmodels and Views.
4.10.1 Navigation: MainViewModel and the ViewModelBase

The different Views are all defined as pages except the main window, which is of course a window. Switching between pages is done through setting the content of the MainWindow to the desired page. To make it easy to switch Views, the content property in the MainWindow has been bound to a property in the MainViewModel. Setting this property triggers the PropertyChanged event and updates the main window with the content corresponding to what that property is set to. The property in MainViewModel is of type ViewModelBase. All view models inherit from ViewModelBase. This means that this property can be set to any view model in the project. A view model is however not a view and does not specify UI logic or design, so in order for the view to change when changing the ViewModel property, the view model needs to be bound to a view. We did this through creating data templates in the app.xaml file.

```xml
<Application.Resources>
    <DataTemplate DataType="{x:Type ViewModels:MainPageViewModel}"
        <Views:MainView />
    </DataTemplate>
    <DataTemplate DataType="{x:Type ViewModels:MakeExerciseViewModel}"
        <Views:MakeExerciseView />
    </DataTemplate>
    <DataTemplate DataType="{x:Type ViewModels:LoadExerciseViewModel}"
        <Views:LoadExerciseView />
    </DataTemplate>
    <DataTemplate DataType="{x:Type ViewModels:RunExerciseViewModel}"
        <Views:RunExerciseView />
    </DataTemplate>
    <DataTemplate DataType="{x:Type ViewModels:FeedbackViewModel}"
        <Views:FeedbackView />
    </DataTemplate>
</Application.Resources>
```

Putting the data templates in this file ensures that the template is shared across the entire application and does not have to be redefined in other XAML-files.

The design we have chosen does not require a central controller responsible for changing pages. Instead it is the individual page, or more precisely the view model that the page is binding to that is responsible for changing from the current page to the next. In a way it’s a form of distributed design where the application flows from one view model to the next, and whichever view model is active, acts as the controller at that point. This is facilitated through a static MainViewModel property in the ViewModelBase class. The property serves as an accessor making the ViewModel property that sets the content of the main window available for all the view models.
protected static MainViewModel mainViewModel;

MainViewModel is used as the data context for the main window. When the main window is initialized it will instantiate an object of MainViewModel. In its own constructor, the MainViewModel will set the static mainViewModel property it is inheriting from ViewModelBase, to itself. Doing this turns the mainViewModel property into a reference to the data context of the main window and ensures that any view model inheriting from ViewModelBase will have access to it. Since MainViewModel is instantiated by the main window as a part of the initialization of the program, and nowhere else, it ensures that any view model trying to access this property, will always be accessing the data context of the main window.

public MainViewModel()
{
    mainViewModel = this;
    ViewModel = new MainPageViewModel();
}

In addition to setting the mainViewModel property to itself, the MainViewModel constructor is also setting a property named ViewModel to an instance of the MainPageViewModel class. This ViewModel property is actually the property that determines the content of the main window.

<Window x:Class="URehabAlphaBeta.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:URehabAlphaBeta"
    xmlns:ViewModels="clr-namespace:URehabAlphaBeta.ViewModels"
    mc:Ignorable="d"
    Content="{Binding Path=ViewModel}"
    Title="MainWindow" Height="400" Width="800">
    <Window.DataContext>
        <ViewModels:MainViewModel/>
    </Window.DataContext>
</Window>

The data context in the main window is explicitly set to ViewModels:MainViewModel, respectively referring to namespace and class. When defined in the XAML like this, an object of the MainViewModel class will be instantiated when the element, in this case the window, is initialized. The other Views does not explicitly set the data context like this. Instead it is implicitly set through the data templates in app.xaml. For instance, MainPageViewModel will automatically be set as the data context to the corresponding view defined in the data template, which in that case is the MainView.
As can be seen by the code above, the binding path of the content attribute, which is determining the property of the binding source, is set to the ViewModel property. The binding source object, since not explicitly set in the binding statement, is determined by the data context. Since the MainPageViewModel is bound to the MainView page, and the ViewModel property is set to the MainPageViewModel in the MainViewModel constructor, the MainView page is displayed as the content of the MainWindow once initialized.

![Diagram showing the initialization chain of mainWindow.](image)

**Program State**

To change from one view to another, the active view model sets the ViewModel property of its static member mainViewModel. This could be done by setting the variable directly to a new object of the desired view model, however, creating a new object each time a page is changed is not suited for keeping the state of the view when changing to a view that has already been visited. If the view has already been visited, one would in most cases want to change the ViewModel property to the existing view model, and not a new view model object. This is done through a method defined in the base class ViewModelBase.

```csharp
protected bool setViewModelByType(Type type)
{
    bool found = false;
    foreach (ViewModelBase vm in ViewModelList)
    {
        if (vm.GetType().Equals(type))
        {
            mainViewModel.ViewModel = vm;
            vm.pageRefocused();
            found = true;
        }
    }
    if (!found)
    {
        mainViewModel.ViewModel = (ViewModelBase)Activator.CreateInstance(type);
    }
    return found;
}
```
The system is designed so that it is only ever requires to have one object of each view model instantiated. This makes it possible to have a method that simply takes the type of the view model and if no object of that type exists, creates a new view model of that type or if it does exist, gets the existing view model and assigns it to the ViewModel property.

The ViewModelList is responsible for keeping the state of each view. It is defined in ViewModelBase as a static list. Each view model is added to the list through a call to the base constructor within its own constructor.

```csharp
public ViewModelBase()
{
    if (!ViewModelBase.ViewModelList.Contains(this))
        ViewModelList.Add(this);
}
```

If the particular view model is found to exist in the ViewModelList, in addition to setting it as the active view model, the PageRefocused method is called. This is a virtual method that is used to execute functionality that needs to be done when a view model is being reused. Since it is a virtual method, this function is defined individually for each view model that has use for it, and can be tailor made to fulfill the needs of any particular view model.

**ViewChain**

Another static list defined in ViewModelBase is the ViewChain. The task of the ViewChain is to keep track of the path traversed through the interface. When a new view model is set, the previous view model is added to the ViewChain. Pressing the go back button in any view will set the content to the last view in the ViewChain and remove that index. This ensures that the last item in the ViewChain will always be the previous page you visited. If the ViewChain is empty, it means that you are at the MainView.

```csharp
protected void goBack()
{
    mainViewModel.ViewModel = ViewChain.Last();
    ViewChain.Remove(ViewChain.Last());
}.
```

### 4.10.2 ViewModels and Views

Common for all the view models is that they take care of all of the presentation logic of the user interface. Code in the code behind files of the views is limited as much as possible. The
only code that is put in the code behind file is invocations of functions in the corresponding view model. This is done through getting the data context of the view and invoking a function on it.

```csharp
private void button_PrepareStart_Click(object sender, RoutedEventArgs e)
{
    ((RunExerciseViewModel)this.DataContext).StartPrepare();
}
```

In the above example the button_PrepareStart_Click function works as an event handler for the button click event of a button in the RunExerciseView. It is, like any other traditional UI event handler, located in the code behind file and it triggers a function called StartPrepare in the RunExerciseViewModel.

This code could be eliminated by using commands. Binding the command property of a user interface element to a command in the view model circumvents the need for an event handler in the code behind file. This is quite normal practice when using MVVM, however it is arguable to which degree this is necessary. Separation between UI logic, presentation and business logic can be done without using commands, and it should not interfere with the testability of the design using unit testing. The use of commands does not reduce the amount of code that has to be written, it could do the opposite, and one could argue whether or not it makes the code any more readable. We made the decision not to use commands because we did not see it as making an important contribution to the software design, and we would rather go with a more familiar approach.

**MakeExerciseViewModel.cs**

This view model is responsible for the creation process of making new exercises. The robot can be moved around and its position is recorded as a waypoint by pressing the set waypoint button. More details about making an exercise and the internal logic of the MakeExerciseViewModel is described in chapter 4.12.1.

**LoadExerciseViewModel.cs**

This view model is responsible for loading exercises from xml files located in the exercise folder and presenting them in the LoadExerciseView. Exercises are listed alphabetically and the list is updated each time the view is opened through the pageRefocused method. Selecting
an exercise in the LoadExerciseView enables the confirm exercise button. Pressing this button activates the RunExerciseViewViewModel and its corresponding view.

**RunExerciseViewViewModel.cs**

RunExerciseViewViewModel is responsible for the presentation logic related to running the selected exercise. The RunExerciseViewViewModel interacts with the ExecuteExercise class, contains the logic for running an exercise. This involves using the exercise data to send commands to the robot, such that the robot performs the intended movements. More details on running an exercise is described in chapter 4.12.2.

### 4.11 Servo Function

The servo function enables the system to adapt to the users input. It does so by measuring the magnitude and direction of the force applied to the handle of the robot. The external force sensor at the base of the handle measures the force. The acquired data is then sent through the USB connection to a computer in a continuous stream. There it is translated to match up with the coordinate system of the robot and subsequently processed together with position and velocity data gathered from the robot. Finally, a command is generated and sent to the command port of the UR5 where it is handled by its internal logic.

There are two implementations of servo functionality in the control software. One implementation adjusts the speed of the TCP along an already defined path based on the amount of force in the direction of motion. The other uses the output of the external force sensor as a parameter for adjusting both direction and speed of the TCP.

#### 4.11.1 Servo in exercise

Performing an exercise with the robot can be done in standard mode and in servo mode. Turning on servo mode adds a dynamic element to the execution of an exercise. While both servo mode and standard mode assist the user in performing an exercise, the servo mode continuously adjusts the speed along the route of the exercise based on force input from the user, giving them direct feedback based on their performance.
Force data

Force data from the external sensor is split in a force component and a torque component, each with an x, y and z component and measured in newton and newton meters respectively. The regular force component measures the force along the axis while the torque component measures the rotational force around the axis. Since we have yet to implement any exercises using rotational motion, the torque component is not used. The three force components can be seen as vectors showing the amount of force along their axis. Adding them together creates a force vector that correspond to the direction of the applied force and whose length correspond to the total amount of force exerted.

In order to adjust the speed along the path of the exercise we are only interested in the component of the force vector that goes in the direction of the path. Taking the dot product of the force vector with the unit vector in the direction of motion yields a new vector with the direction of motion and the magnitude of the force in that direction. The position tangent in the RobotData class serves as the direction vector and normalizing this vector gives the unit vector in that direction.
The speed is changed by a scalar value of the force in the direction of motion. It is done through continuously sending updated commands to the robot. The update commands contain the same destination point as the previous command and only change the value of the velocity.

Figure 39 Showing sequence of command updates and their origin.

While determined by the magnitude of the force in the direction of motion, the velocity is restricted by a maximum value to ensure that operational conditions are safe. For the exercises, a lower limit has also been set.

4.11.2 Free Motion Servo

We have also implemented a free motion servo system where the movement of the robot is entirely based on input from the user. The free motion servo differs from the exercise servo mode in that it does not calculate force in the direction of motion. Instead it calculates motion in the direction of force. This is done by translating the force vector to the coordinate system of the robot and adding the force vector to the current position of the TCP. The result gives a vector whose endpoint is a point in the direction of the applied force in the coordinate system of the robot. This is then scaled by a factor that keeps it within the working space of the robot while not being so close to the current position that the TCP can reach it before the next command is processed.
As with the exercise servo mode, the velocity is calculated as a scalar value of the force. Unlike the exercise variant however, the free motion servo also changes the acceleration. This is based on a different scalar of the same force. The choice of scalar values is based on what we felt was a good balance between smoothness of motion, responsiveness and speed.

While the core function of the free motion servo is a straight translation of the force vector, different conditions are handled differently in order to make a smoother and more usable servo function. For instance, a considerable challenge when using force data directly to control the motion is that sudden changes in direction also impacts the force sensor which subsequently impacts the motion again. This means that the force control opens up a feedback loop where each change in motion can trigger another motion in the opposite direction without any additional user input. This is one of the first issues we faced when prototyping the servo function. In fact the robot could at times get totally out of hand if the changes in force and direction were too sudden. To deal with this issue we check the angle between the latest force vector gathered from the force sensor, with the previous force vector. If the angle is too big, instead of sending a new movement command, a stop command is sent. In addition to avoiding the feedback loop, this also makes sense if the user intends to move in the opposite direction because the stop command decelerates the motion before the direction is changed making for a smoother movement.

```csharp
if (Vector3D.AngleBetween(previousForceVector, forceV) > 40)
{
    command = String.Format("stop(a={0})\n", stopAcceleration.ToString(CultureInfo.GetCultureInfo("en-GB")));
    stopped = true;
}
```
As can be seen above the command property is set to a linear stop command with a deceleration value contained in the stopAcceleration variable. The boolean value stopped, which here is set to true, is used to flag that the robot is in the process of stopping. If in the next update cycle, the robot has a velocity higher than a threshold value, the command will be set to stop once more.

```csharp
if (stopped && Robot.CurrentData.TcpSpeed > 0.10)
{
    command = String.Format("stopl(a=1)\n");
    Robot.SendCommand(command);
}
```

Doing this ensures that the TCP goes no faster than 0.10 m/s before making changes in direction greater than 40 degrees in a cone radius around the previous direction.

![Figure 41 Showing the cone used for deciding sharp direction changes.](image)

While 90 degrees would be enough to take care of most of the feedback problem, a 40-degree angle of acceptance was chosen in order to avoid very abrupt changes in direction at high velocities. Another measure is to take any change in force direction between 40 degrees and 10 degrees and create a new movement direction that is half that angle away from the previous direction. This gives the TCP some momentum in the direction it is traveling and it can at most change direction by 20 degrees between each update at high velocities. Finally, a threshold value sets a lower bound on the amount of force needed to move the robot. This value is determined by a constant multiplied with a friction variable that can be adjusted to change the robot’s sensitivity to force input.

The force vector used in calculating the moves for the robot in the free motion servo class is an average of the last ten force vectors received rather than just the latest. This makes the force vector less affected by small changes, noise and short spikes in force. Since the force sensor updates at 100hz, it takes 100 milliseconds to replace all of the force data that
contribute to creating the averaged force vector. The frequency of updates to the robot is set to 70 milliseconds. This means that a component of the previous force vector will be present in the next, adding to the continuity of the motion. Using averaged data like this adds a delay proportional to the number of averaged elements meaning that there is a tradeoff to be made between having a better sample base and a more responsive design.

The starting and stopping acceleration is adjusted based on the magnitude of the force vector. Making this adjustment allows for very smooth motions when the force input is small, because a low acceleration helps blend the subsequent movements together. If the force input is greater, the greater acceleration ensures that movements still feels quick and responsive.

**Considerations about servo functionality**

While a number of measures has been taken to create a smooth and functional servo experience, we do not claim that this is an optimal solution. Several steps could be made to further improve this functionality. First of all, additional measures could be taken to negate feedback in the force sensor when the robot change speed and direction. For instance, we do not calculate the amount of force caused by the inertia of the handle when changing speed or direction. Negating this force should remove much of the issue with the feedback loop. Another problem is that change in speed will cause a change in force from the user because of the inertia of the user’s arm. This problem is exacerbated by the delay from user input to motion response. Trying to calculate the inertia of the user’s arm could mitigate part of the issue, but it does not solve for corrective behavior on the part of the user based on the expectation of instant feedback.

The ideal solution to this would be a significantly lower latency from force input to execution of movement. The update frequency adds a latency from 0 to 70 milliseconds and the averaging of force data also has a minor impact on responsiveness, but the main contributing factor to the latency is the robot’s internal calculation and initiation of movement. This has reportedly been improved in the latest beta firmware [51]. A way to mitigate this issue without reducing the latency is by adding a momentum component in the control logic. Doing this you can simulate inertia on the robot arm itself. This simulated inertia requires that a sufficient amount of energy is used in order to stop or turn the robot’s direction, rather than just any force in another direction. Although it is desirable that moving the TCP require as
little energy as possible, adding a momentum factor is likely to improve the servo, making it work more intuitively.

The update frequency of the servo functions is based on the rate of which the robot is able to execute subsequent commands. We measured this to be around 60 to 65 milliseconds described in chapter 3.2.3. A possible optimization could be to have a variable update rate rather than a fixed one. It would still need at least a 70 millisecond pause between each command, but skipping unnecessary commands, like when the next command is very similar to the previous command, it can react more quickly when a change it finds to be worthwhile occur. Of course when performing motions where every subsequent command is deemed significant it will have the same 70 millisecond update frequency.

4.11.1 Tracker Module

Controlling the robot on a movement by movement basis poses a challenge. As shown in chapter 3.2.3, sending several commands simultaneously only causes the last command to be executed. This means that the next movement has to be sent as the previous movement finishes. Unfortunately, the robot does not return any message telling that the movement is complete.

In order to know when a movement finishes, we built a tracker module that monitors the position of the TCP as the robot is executing a movement. As the movement approaches its endpoint, the tracker executes an event signaling that the movement is nearing completion. Since there are inaccuracies involved when working with positions in 3D space, the tracker is given a tracking-radius that determines whether or not the current position is acceptably close to the endpoint of the movement. If the endpoint gets within the tracking-radius, the event is triggered, signaling that new movement commands can be sent.
In order to ensure continuous movement between commands we also have to take into
consideration the delay discussed in chapter 3.2, between the robot receiving a command and
executing it. If a new command is sent the same moment as the previous command finishes,
the robot will stop at the endpoint of the previous command for a short time, corresponding to
the delay. This means that the tracker has to signal some time before the endpoint is reached if
we are to avoid stopping between subsequent commands. To solve for this, we calibrated the
radius of the tracking module such that it signaled early enough for the robot to start
executing a new movement before stopping, but not so early that it would cut the previous
movement short.
Using a radius to determine if the robot is close enough to the endpoint will not yield the same result for all movements. A linear movement will have a shorter distance left when the endpoint enters the radius than a curved movement, as a direct line is always the shortest path to the target. From what we have observed, this difference does not appear to have a significant impact for the types of movements that has been tested. Another issue arises when the tracker is combined with the servo mode. Using the servo mode, the speed of the TCP can vary over the course of the movement. When the TCP has a higher speed, the endpoint will be reached sooner, so in order to avoid stopping at the end point, the tracking radius needs to be bigger. We try to solve this by having a dynamic tracking radius that changes based on the velocity of the TCP. This appears to work well, but there are thinkable scenarios like if the TCP changes velocity very rapidly from a high speed to a low speed just as it is entering the tracking radius, that the movement could be cut a little short.

Figure 44 Sequence diagram showing operation of the tracker.

Figure 44 Sequence diagram showing operation of the tracker. shows the tracker being used in the context of executing an exercise. It shows the communication between the different components involved in the process of tracking the position and destination point.

**4.12 Designing the Interactions**

This chapter will describe the design and logic of the user interactions throughout the system. The main use cases will be presented and choices made in development and design will be discussed.
4.12.1 Making an Exercise

One of the features of the software is the ability to create exercises. It allows medical personnel and people qualified in biomechanics and physical therapy to create the exercises and gives them the opportunity of making personalized exercises for the individual user. Recognizing that our field of expertise is not within biomechanics or physical rehabilitation, we have mainly been focusing on the efficiency of the creation process and the general functionality, rather than specific functionality based on detailed knowledge from those fields. We have tried to condense the interactions required to create an exercise down to as few steps as possible, and focus on the core functionality of the exercises.

When making the tool for creating exercises, we thought about the possibility of tracing the movement of the robot and “record” it as an exercise. A good thing about this approach is that it could be very efficient. If you just wanted a simple exercise you could just press “record”, then move the robot, and press finish. You would also know exactly how the robot would move, because it would be the same path as you traced out. There are however a few potential problems with this approach. First, tracing the exercise might not be that simple if you are trying to be accurate. We figured it likely that it would be difficult to create a smooth exercise this way without processing the position data through a smoothness algorithm. Using that approach, it would have to be an approximation of what the user intended. Another issue is that it would require many data points, at least if we wanted to make a good approximation, which we figured could make it hard to work with during implementation, specifically as it relates to the servo function described in chapter 4.10.

What we decided to do was to part the exercise into several movements and let the user decide the endpoint of each movement. This way the user has full control over the positions of all the points in the exercise, and only points important to the exercise is likely to be registered. The user can decide between linear movements and curved movements, taking advantage of the robots movep and movec functions respectively talked about in chapter 4.5.2. A linear movement only require an endpoint in order to make a line from the current position to the point. A curved movement requires a midpoint as well as an endpoint in order to calculate a curve from the starting position, through the midpoint, and to the endpoint. This follows from the geometrical law that three points in space define a circle. Another benefit of this approach is that it makes it easier to adjust segments of the exercise individually, allowing features to be added or specific changes to be made to each movement. In our implementation, we
included the ability to add waiting times for each movement, setting the time it would take from finishing that movement until the next movement initiates.

While it would be possible to use the robot’s free drive mode to position the robot when selecting the endpoints for the movements of the exercise, considering safety flaws discussed in 4.1.1, we decided that it would be better to use the free movement servo function that we have developed. There are a couple of reasons for this. First, taking advantage of the robots own motors makes it easier to move the robot from position to position. Secondly, the free motion servo operates with the same fixed rotational position of the TCP that is used during exercises. This means that the position will be the same when creating the exercise as when performing it and it also means that it is not possible to add points that the robot cannot reach during execution of the exercise. Finally, it avoids issues related to releasing the breaks on the robot when entering free drive mode. If the payload has not been correctly calibrated in the initialization of the robot, releasing the breaks poses a safety risk as the robot arm could drop towards the ground. If fully extended it could potentially reach a high velocity, posing a risk to the wellbeing of the patient. When using the free drive mode, it will resist any force deviating from its path, and will go in to emergency stop if the security force threshold is surpassed.
In Figure 45 above, the workflow of the exercise creation process is shown. The process starts by choosing a movement type. After selecting the movement type, the robot is positioned and the position is stored as a point in the exercise. For a linear movement this point will be the endpoint of that specific movement. For a curved movement that will be the midpoint of that movement and another position is then chosen as the endpoint. A waiting time is then added,
corresponding to the amount of seconds it should wait after that particular movement. Setting this value to zero corresponds to no waiting time. This process is then repeated for as long as the user wishes to add movements to the exercise. When the user has added all the intended movements to the exercise, it will be named and put through a test run before it is successfully created.

The purpose of the test run is to ensure that the exercise works as intended do some configurations to allow it to work better with the exercise servo. The test run is done by creating an exercise object from the data retrieved from the creation process and using the ExecuteExercise module to perform the exercise. The exercise is configured by adding a direction vector for each movement and by changing the midpoint of each curved movement to a position close to the endpoint. The direction vector is used as the initial position tangent each time a new movement command is sent to the robot. The position tangent is used to calculate the magnitude of force that is applied in the direction of motion and is used in the exercise servo mode. Putting the midpoint close to the endpoint is needed for the servo to work properly. The reason for this is that it continuously needs to update the speed of the same movement, containing the same endpoints. When the robot goes past the midpoint, resending the same command will send the robot in the opposite direction, or cause an infinite radius exception caused by the robot being unable to calculate a circular movement through the midpoint to the end point. After finishing the test run, the exercise will be written to an xml file and stored.
Figure 46 Sequence diagram showing creation of exercise.

Figure 46 Sequence diagram showing creation of exercise shows the operational sequence of making an exercise. The scope is limited in order to fit the most relevant operations. FreeMove operational sequence is explained in more detail in chapter 4.11 on servo functionality. The internal operational sequence of the test and calibration run is not explained but it follows much of the same operations as running an exercise, which is shown in its own diagram below (Figure 47).

4.12.2 Running an Exercise

As with the creation process, we wanted to keep the interactions required to run an exercise at a minimum. We identified the required steps necessary to perform the exercise. First, the robot needs to be positioned in the starting position of the exercise. Secondly, the force sensor
needs to be calibrated to adapt to the weight of the user’s arm. After performing these initial steps, the exercise is ready to start.

Upon starting the exercise, the nextMove function is called. This function checks that there are movements left to be performed before sending the latest move to the robot. The tracker is used to notify when the next movement can be sent to the robot. The endpoint of the movement is sent to the trackers SetPointToTrack function. When the robot's position is sufficiently close to the endpoint, the trackers event is triggered and the nextMove function is
called again. This sequence repeats itself for as long as there are more movements left to be performed. When all the movement in the exercise is performed, the nextMove function will check if any repetitions has been set. If there are more repetitions to be performed, the exercise is set to perform in reverse and the repetition counter is decremented by one. Now the nextMove function will start at the last movement iterating down towards the first movement as the robot progresses from the final endpoint to the starting point of the exercise. A few modifications are made for the movement to work correctly when performed in the opposite direction. If the movement is curved, the original midpoint is used to create the direction vector of the reversed movement, and the original direction vector is used to create the new midpoint. The endpoint of any reversed movement is set to the endpoint of the preceding move. When the robot reaches its original starting position, the exercise is performed once more in the normal direction. This process repeats itself for as long as the repetition counter remain higher than one.
Figure 48 shows the sequence diagram of executing an exercise. Its scope is limited to the internal operation of the computer software and avoids details about the communication with the robot. In addition to sending the sequence of commands from the exercise, a parallel process is responsible for updating the velocity of the exercise. This process is responsible for the servo functionality of the exercise, adjusting the speed according to the force input from the user. It works independently from the rest of the execution logic, updating the speed based on its own timer. This timer, as described in chapter 4.11 runs at a frequency corresponding to results from the update frequency tests of the robot in chapter 3.2.3.
4.12.3 UI Design

The following chapter describes the design of the graphical user interface. It includes a short description of each view and discusses the different design decisions.

Home Screen

It has been our goal to make the interaction with the software as straightforward as possible. The home screen embodies this by having only two options available for the regular user. One option is opening the game mode, and the other is starting an exercise.

![Figure 49 Screenshot of the home screen of the application.](image)

The buttons and text are made big so that it is easy to hit the buttons and easy to read what they say. The two user buttons are placed in the top left quadrant of the screen matching with how people are used to read information, from top left to bottom right. Two additional buttons are placed in the bottom left quadrant and is unavailable unless the advanced mode checkbox is checked. It is kept in sight so that users know that it is there, but they are made transparent in order to give visible feedback that they are disabled. These functions are intended for medical personnel like physical therapists or doctors.
Create Exercise Interface

MakeExerciseView is the interface used for creating new exercises for the user. The main goal of the interface has been making the interactions as simple and efficient as possible. In order to add to the visibility of the interface, all steps required in making the exercise are visible on screen at the same time. The interface has only a few steps adding to the clarity of the interface. In addition, only one section is highlighted at a time, while the rest are transparent. The highlighted section is the one the user currently needs to address and it helps direct the attention to what is important. The transparent sections are made unavailable to make sure the user cannot perform unwanted actions. The sections are placed in order of usage from top to bottom following the logical path the user would expect. An information bracket exist in the bottom left quadrant to guide new users in creating an exercise.

Figure 50 Screenshot from the exercise creation screen in the application.

First section is about choosing the movement type and setting the endpoint, or in the case of a curved movement, the midpoint and endpoint for the movement. The robot is to be moved to the desired position before the save waypoint button is pressed. In the case of a curved move, both the midpoint and endpoint needs to be set before the next section is enabled. The second section is for adding a waiting time between movements. This is set by a simple slider to mitigate risk of making a mistake and at the same time avoiding the need for input validation. The third and final section is responsible for naming and creating the exercise. The user can add more points by pressing the yes button, which will reactivate the first section, starting
again from the top. If not, pressing no will make the create exercise button available. Upon creating the exercise and completing the test run, the user is greeted with a display showing that the exercise was successfully completed.

**LoadExerciseView**

This view is responsible for displaying the different exercises that is available for the user. The exercises are loaded from xml files and displayed dynamically. As can be seen in Figure 51 up to three exercises can be listed horizontally before continuing on the next row. The view uses the same consistent visuals for displaying currently unavailable actions by making them transparent. It follows the same visual design as the other views and the same approach to a simple design with few options, big buttons and big text.

![Select an Exercise](image)

*Figure 51 Screenshot from the exercise selection screen in the application.*

**RunExerciseView**

This is the view presented to the user when they have chosen an exercise to perform. The process of running an exercise follows a linear path containing a few simple interactions. We found it best to use a single button for these interactions as no more is required. This way,
there is only one path through the interactions and very little room to make mistakes.

![Figure 52 Screenshot from initial state of executing an exercise.](image1)

The button is first used for the preparation step. This gives the user control of when the robot should be moved into the starting position. This way it does not move without the user knowing, thereby mitigating the risk of the user getting hit by the robot, or scared because it moved unexpectedly.

![Figure 53 Screenshot from calibrating force sensor as second step of executing an exercise.](image2)
The button is also used to when calibrating the force sensor. It is important that the user is holding the handle when the calibration takes place, and using the button for this allows the user to validate that the calibration is done correctly.

Finally, the button is used for starting an exercise, and for running an exercise one more time after it finishes. In addition to the button, there is a slider to adjust the amount of repetitions to be performed, and another slider at the bottom setting the sensitivity of the servo function.

Figure 54 Screenshot from starting position of executing an exercise.

Figure 55 Screenshot showing the force gauge during exercise execution.
In addition, there is a visual feedback to the force input from the user through a green power-
bar filling the background of the control box. It follows the same visual design as the rest of
the views.

4.13 Gamification in this project

As described in chapter 2.7, gamification is added for increased user involvement and
motivation. The principles of gamification have been part of planning throughout this project.
In the current prototype of the solution developed, a proof of concept-game is added in the
application. Based on the idea of turning tedious tasks into games, the robot arm is turned into
a game controller. The thought behind this is turning the task of physical training into a game.

Although the current version of the software has a limited implementation of gamification, it
was part of choices made in development, and a framework for adding games on top of the
application is present. The modules for tracking position, timer and freemove servo function
are all accessible for game controls.

![Figure 56 Showing the racing game implemented as gamification.](image)

The current implementation of gamification is in the form of a car racing game with
possibilities for different tracks. A user controls the car by moving the robot arm in the
direction desired. Based on force issued by the user, both robot and car speed is changed. If
the car crashes with the edge of the track, both the car and robot stops. The user will have to move away from the edge before moving further. Game elements included are tracking of speed and top speed, as well as lap times.

**Further Implementation of Gamification**

Depending on user classification, different and additional functionality and game elements should be opened up. This could be according to therapy plans, abilities or interests of the users. Using the principles of modular extensible design will help ensure a composure suitable for the specific user at the time. These could include different levels of sophistication in game environments and objectives, as well as wider ranging and more complex reward system. There should also be differentiated forms of audio-visual feedback depending on the current user.

Simple action-response functionality should be implemented as an entry level, or less advanced form of game-interaction with the system. This could be done by giving the user feedback as a response to physical interaction with the system controller. A realization of this could be a hand tickling a stuffed animal, using pins to pop balloons, or some other form of low complexity interaction. At this level of relatively low advancement, the gamification principles used would also have to be limited. Rewarding users more directly with clear visual or audial stimuli, will be fundamental for user engagement. An implementation of this could be using shooting stars, bouncing balls, and fanfares to encourage and excite.

As users’ physical ability, mental capability, experience with the system, and desired movement in the exercises increase, the level of advancement should also go up. At this point, the interaction could entail steering a car down a track (as implemented in this project), leading an object through a maze, or simple ball games like table tennis or air hockey. Now the introduction of additional game elements will be natural. Adding abilities to level up, choosing difficulty of opponents or environments, and earning upgrades or items along the way.
5 User System Evaluation

Although viability of the core concepts of the project is established through earlier research discussed in chapter 2, we would like to survey initial thought from potential users. As one of the central goals of the project is developing a system with potential for a high adoption rate, the user experience and overall perception of the system is important. There are two actual user groups for the system developed; physical rehabilitation patients and health professionals. Since the first group contains persons in a particularly vulnerable situation, it was decided that the system in the current development state would not be tested on actual patients. There are in addition strict regulations for patient testing, and it was deemed impractical in relation to the quality increase in data acquired.

The user group proposed for user testing is ergo therapy students. This group will have some background from rehabilitation work, and in particular some experience with the user group of patients.

Some central points are proposed as the criteria for user testing:

- **Thoughts and Initial Evaluation of the Principles: (PRINCIPLE)**
  Using robot assisted therapy as a supplement in treatment. The entire system as it is, has a substantial size and presence in the room. It can be perceived as a fairly mechanical and cold installation.
  - How do testers react to this, and are any improvements suggested?

- **Initial Operation of the System: (USER EXPERIENCE)**
  In the current form, the system consists the robot box, a PC and two cables which need connecting. The power button of the robot must be pressed, and a desktop application must be started.
  - Is the solution sufficiently user friendly in initial operation?

- **Exercise Creation and Execution: (USER EXPERIENCE)**
  Creating exercises is done through the application and moving the robot manually. After creation, exercises can be loaded and executed from the application.
  - Do the testers find the principles of creating and executing exercises fitting?

- **Physical Implementation: (PHYSICAL IMPLEMENTATION)**
  The robot is mounted on a table with wheels. The handle is static.
  - Are there any suggestions on further development for the physical implementation?
Final Impressions and Thoughts: (CONCLUDING THOUGHTS)
After trying the different user scenarios, final thoughts and impressions will be noted. Some points will be surveyed.
  o Approachability of such a system
  o Ease of use
  o Work flow when using such a system
  o Shortcomings and improvements

User Evaluation Ergo Therapy Students

The group of ergo therapy students were chosen as representatives from the health professional user group. Consisting of 6 students, the individuals were subjected to one 20-30-minute session with the system each. The sessions started with an initial conversation about their thoughts about the principle of robot assisted physical therapy. Both positives and negatives of using robots in treatment were discussed. In addition, the introduction of robot treatment into homes were presented to the subject for comment.

Following the initial conversation, the subjects were asked to start up the system and comment along the way. The robot was already started, as a cold start takes several minutes. The implementation of control from a PC application was discussed, as well as the process of connecting and starting the system.

The next step in the evaluation was running through the central use cases of the system. Subject were encouraged to try on their own, and ask if anything was unclear. They were also asked for general comments along the way. Three use cases were conducted:

Creating an exercise:
Enter the exercise creation, select desired points, time delays and movement types to construct an exercise. Save exercise.

Load and Run Exercise
Enter exercise load screen and select an exercise. Follow directions and complete the selected exercise.

Try Gamification
Select the “Let’s Play” button. Run through an instance of the game.
Throughout the operations, subjects were encouraged to comment on their thoughts and actions and give feedback on the system.

After completion of the different use cases, a conversation around physical implementation of this system in particular, as well as similar system in general was had. Subjects were encouraged to give thoughts on the different parts of the system, and especially what could be improved. The principles of exercises as implemented was also discussed.

The system was introduced as a prototype with several technical shortcomings. Most notably the fact that the controller software running on the robot freezes and crashes from time to time. As this is a known problem and is thought to be corrected with future software updates, the subjects were asked to overlook this during testing.

5.1.1 Results from User Testing

The results from ergo therapy user testing will be presented in the following chapters. Results are divided into the categories principles, user experience, physical implementation and concluding thoughts.

Results: Principles

All subjects expressed both enthusiasm and skepticism towards the use of robots in physical rehabilitation treatment. Firstly, a concern of robots completing movements for patients was expressed. This was explained by the need for sufficient muscle activation in users for desired training results. However, for some users not able to move on their own, movement completed by the robot could be beneficial. When introduced to the addition of the dynamic servo function, that particular concern was diminished. The idea of the robot ranging from completing movement for a user, through guiding and aiding if necessary, to resisting in movement was seen as beneficial.

Further, the need for properly administered training programs for patients was stressed. Different users need different training. The idea of making unique exercises on a user to user basis was seen as a good approach. All subjects mentioned the importance of not giving users what they called “institutional feel”. This applied to both the physical implementation and the software.
Adoption of a system based on robot assisted physical therapy was also seen as a challenge. From experience from work, one subject in particular pointed out big individual differences in acceptance of aids in general, and technologically advanced aids in particular. This view was shared by several other subjects as well. It was noted that there could be big differences between demographic groups, and that elderly users probably would be most difficult to introduce to such a system. On the other hand, it was thought that for some groups, it would be exiting and motivating in itself to try and use new technology.

The views on introducing this kind of system into the homes of users differed somewhat between the subject. Some saw it as a good way of offering more therapy to users. Others noted that it could reduce the social interaction of users not having to leave the home for training, and that this social interaction was crucial for some users. Several subjects pointed out that this should be an option available for choice by user and health personnel in conjunction.

**Results: User Experience**

First the installation, connection and initial operation of the system was discussed. There were no objections to the amount of wires or connections, as this was seen as a one-time installation process. There was however expressed a desire for controlling from a touch enabled tablet for simpler interaction. The possibility of wireless connection was also mentioned.

When starting the application, the simplicity of the graphical user interface was appreciated. The limited amount of choices and actions needed was seen as a good design choice. Some instructions were needed throughout operation. This was mostly needed for the relatively advanced tasks in exercise creation. A wish for more tooltips through the processes was expressed. Especially curved movement creation was challenging given limited training. The use of more contrasting colors and larger fonts was encouraged.

While executing an exercise, the wish for better progress reporting was stated. Both during a single exercise, and on a session to session basis. This was also proposed as a way of motivating users.

Moreover, all subject reacted positively to the addition of game elements trough gamification. Enthusiasm were shown while trying the concept implemented trough a simple car racing
game controlled via the robot arm. The importance of different games offered for different users were stressed. Several subject pointed out that for some users, games would not be beneficial, but rather make them hostile to the system. Again the big individual differences between users were noted.

It was further commented that the physical implementation of the system played a part in the user experience. Again the importance of avoiding the “institutional feel” was stressed. Both the appearance of the robot and the handle was posed as central points.

Results: Physical Implementation

As mentioned as part of the user experience results, both the appearance and the handle were seen as significant by all the subject. Firstly, the robot as it is, was seen as cold and uninviting. Proposals of covering the metal in fabric, and hiding all wires were uttered. The fairly substantial table used as a base for the robot was also commented. It was proposed to either mount the robot on a smaller and more mobile base, or potentially mount it directly on a stationary table or on the wall.

The handle was focused on for several reasons. First, the handle does not swing or turn to accommodate different angles throughout movements of the robot. This could lead to unnatural joint positions for users. It was noted that protection of joints was vital in physical therapy. Furthermore, the ability to grip could differ a lot form user to user. Especially with users suffering from lowered functionality in their arms. The possibility of interchangeable handles was seen with enthusiasm. Both straps and glove like handles were discussed.

Another point made was the limitation in range of the robot. For some exercises, it could prove difficult to find paths the robot could follow. The focus on upper limbs were also commented. This was seen as good in the sense of upper body strength training, but a wish for addition of lower limbs was stated.

Results: Concluding Thoughts

The concluding thought from the subject were similar for the most part. Robots used in physical therapy was seen as a good addition, but not a substitute of traditional therapy. The importance of not removing the social aspects from therapy was stressed again.

120
Furthermore, finding a better physical implementation was seen as necessary. Making the system as non-intrusive as possible was noted as important. It was also said that big individual differences between user needed to be considered throughout the development process.

The minimalist GUI was seen as effective, but a need for better contrast and bigger fonts were seen. Big buttons and few clicks was appreciated. In addition, the inclusion of gamification was again seen with enthusiasm.

### 5.1.2 Conclusion from User Testing

Before concluding, some clarification on the test are needed. Firstly, the user testing and evaluation did not adhere to any strict standards for user testing. The user testing conducted is seen as a channel for feedback for use in potential future work and development. It will therefore not be used for definitive validation of the system, but rather as a way of pinpointing potential problem areas.

From the results of user testing we extract that there is probably both positives and negatives of using robots in physical therapy. The opinion in our case seems to be it is a good addition to traditional therapy when used correctly and with care. It seems the idea of smaller, cheaper and more mobile solutions than the ones available on the market could be popular.

It is seen as necessary to focus on the physical implementation in future development to ensure better quality of physical therapy, as well as user acceptance. This part of development should be done in conjunction with professionals of the specific field.

The inclusion of gamification seems to show promise, and it is seen as worthwhile to continue development on this aspect. Possibilities for different games are seen as necessary to cater for a bigger number of users.

In the case of user interface, there has not been any clear problems with the minimalist approach, but there should probably be more thought going in to accessibility for users with disabilities.
5.1.3 Heuristic Assessment by the Authors

In this chapter an assessment according to the heuristics described in chapter 272.6 is conducted. This is conducted as an evaluation of the user experience from the perspective of the developers. Both positives and negatives will be presented, as well as some discussion around the choices made. The assessment is based on our own evaluation, as well as feedback gotten through user testing.

Visibility of System Status:

There is not any global indication of where in the program navigation structure the user is. This could be done, but was chosen not to implement at it was seen as adding additional clutter to the navigation. As it is, the system is no more than two levels deep in navigation, so it was decided to not include this.

In the processes of creating and executing exercises, system status is continuously shown through text, focus and visibility changing depending on the step in the process. The visibility of system status is seen as satisfactory.

Match between system and real world:

To aid the match between system and real world, naming has been decided to follow general names used outside the application. An exercise is called an exercise, and all explanations attempts to keep language generic.

User Control and freedom:

The application offers the choice to go back to home from any page. System states are kept so that a user can go back and forward as many times as desired without losing anything.

Where the application does lack in user control and freedom is in the exercise creation process. Ideally the user should be able to undo one step, and not have to start again if a mistake is done. Additionally, in running an exercise, the possibility of restarting an exercise before completion and without going back to start and choosing the exercise again, should be available.
**Consistency and standards:**

To keep the user experience consistent, the same names are used in all different parts of the system. In addition, global design styles are implemented so that a button looks like another button wherever the user is in the application.

**Error Prevention:**

Throughout the development the concept of error prevention has been considered continuously. The approach of keeping design as minimal as possible and to keep the workflows short was part of the error prevention. In the use cases of a patient, the error prevention is seen to be fairly good.

However, in the exercise creation process, there are some possibilities for error. Especially when creating a curved movement as part of an exercise. Such a movement is created by inputting two points on the circumference of an arch. It is however possible to input the same point twice, resulting in an error. In addition, when creating curved movements, it is possible to construct a path the robot is unable to follow.

**Recognition rather than recall:**

The application is designed so that all choices are done via buttons or a presentation listing all the options. Exercises available are shown with name allowing the user to recognize the appropriate exercise.

**Flexibility and efficiency of use:**

In development, the need for shortcuts in the application was not seen as necessary due to the relatively flat architecture of the system. The different use cases and functions available are one action away from the home screen.

One functionality potentially increasing efficiency for users is the ability to add exercises by copying the corresponding XML-files into an exercise folder in the application directory. In the cases where the same exercises can be used across different users, this saves an operator for having to input the exercises on each device.
Aesthetics and minimalist design:

It was decided from the start that a minimalist design should be implemented to make use of the system as simple as possible. This has resulted in few objects on every screen, and limited options for the user to choose from.

The color palette of the GUI has been chosen with the intention of being non-intrusive and pleasing aesthetically. There has also been a focus on universal access since some users are likely to need this. Making fonts big enough to read, using contrasting colors, and making interaction objects large enough for users to use.

Even more work on making the application accessible could be done. Even bigger fonts and interaction objects should be implemented.

Help users recognize, diagnose and recover from errors:

The application should ideally contain more tooltips throughout operation. A user should in addition get feedback along the way to keep them more informed about what the system is doing.

In the current state of the system, with software crashes in the robot, the application should incorporate a reset connection option instead of having to restart the application. This is however a problem which should disappear when the robot software receives further updates from the manufacturer.

Help and documentation:

The addition of a global “?”-button could help the user in using the system. This should be implemented. In addition, the tooltips throughout the system could be improved.
6 Conclusions

Throughout this master thesis we have sought to establish the viability of developing cheaper, mobile and user friendly systems for robot assisted physical therapy. The software controlling such a system has been weighted heavily. This chapter will offer conclusions drawn from the results gathered.

6.1.1 Concluding the Principle of Robot Assisted Physical Therapy

Through a review of previous research done on the subject of RAPT, the viability of the concept has been established. There seems to be an agreement around the value of RAPT as an addition to traditional therapy. We would recommend further work in the field of smaller mobile solutions.

6.1.2 Concluding the Technical Platform

In this thesis the technical platform of the UR5 with the added FT-150 force sensor has been shown to meet the requirements set for the system. With further development the problems noted are believed to diminish.

Additionally, the safety features of the UR5 makes it fit for the task of RAPT. The physical size and weight is well suited for mobile installations.

6.1.3 Concluding the Prototype

After user evaluation we conclude the solution does show promise. Especially the concept of added gamification is valuable. The prototype is still in a development phase, and should be further developed. The robot assistance ranging from completing unfinished movements to resisting movements is seen as the right approach.

6.1.4 Future Work and Improvements

This thesis has aimed to first establish viability of the principles of RAPT and expanding RAPT to the home. Further, the main part has been developing a prototype to show the principles in practice. As it stands, an operative prototype is present with core functionality in place.
As shown in chapter 3, the technical platform does perform satisfactory, although some improvements would be desired. New software updates of the control software should be evaluated. Further work should seek to work closer with the manufacturer of the robot to ensure tighter integration. In addition, work could be done with the sibling models of the UR5; the UR3 and UR10.

The physical implementation should be improved by reducing size of the installation. Further, the possibility of switching communication to wireless and including a tablet for the control application could be implemented. A collaborative project with health professionals to enhance the biomechanical aspects and therapy results could be conducted.

Another aspect we see promise in developing further is gamification. Adding more game elements and building a more complete experience. As it stands, the gamification on top is a standalone game integrated directly. The possibility of integrating game engines like Unity or Unreal could greatly improve the experience, and should be investigated in further research.

6.2 Final Conclusions from the Authors

This thesis has presented the research conducted as well as the prototype developed. It is of our opinion that the results of this work is worth further research, and that the principles laid ground are validated.

The decision to go down the route travelled in this project was done on the basis of what was available in terms of equipment. As the research in this project is conducted by request from HiB, the choice of platform fell on the UR5 as this was available. Using the Universal Robots platform is viable, but this approach has not been extensively evaluated against other potential solutions.

We as authors see the gamification aspect of the implementation as the most interesting and strongly believe that this has a lot of potential. Specifically, in aiding motivation of users.

While the prototype implemented in this thesis might never be used on patients, we believe that the concepts bear value outside this specific solution. The future of healthcare as we see it involves robotics in one form or another, and the principles researched throughout this project will be a part of that future.
References


Appendix

1. Universal Robots UR5 Specsheet
2. Robotiq FT-150 Force Sensor Specsheet
3. Data from Delay – to Moving Experiment
4. Data from Delay – to Stationary Experiment
5. Source Code
# UR5 Technical specifications

**Item no. 110105**

## UR5 6-axis robot arm with a working radius of 850 mm / 33.5 in

<table>
<thead>
<tr>
<th><strong>Weight:</strong></th>
<th>18.4 kg / 40.6 lbs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Payload:</strong></td>
<td>5 kg / 11 lbs</td>
</tr>
<tr>
<td><strong>Reach:</strong></td>
<td>850 mm / 33.5 in</td>
</tr>
<tr>
<td><strong>Joint range:</strong></td>
<td>$\pm 360^\circ$</td>
</tr>
<tr>
<td><strong>Speed:</strong></td>
<td>All joints: 1800°/s. Tool: Typical 1 m/s / 39.4 in/s.</td>
</tr>
<tr>
<td><strong>Repeatability:</strong></td>
<td>$\pm 0.1$ mm / $\pm 0.0039$ in (4 mils)</td>
</tr>
<tr>
<td><strong>Footprint:</strong></td>
<td>Ø149 mm / 5.9 in</td>
</tr>
<tr>
<td><strong>Degrees of freedom:</strong></td>
<td>6 rotating joints</td>
</tr>
<tr>
<td><strong>Control box size (WxHxD):</strong></td>
<td>475 mm x 243 mm x 268 mm / 18.7 x 9.6 x 10.6 in</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>I/O ports:</strong></th>
<th>Control box</th>
<th>Tool conn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital in</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>Digital out</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>Analog in</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Analog out</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>

| **U/O power supply:** | 24 V / 2A in control box and 12 V / 24 V 600 mA in tool |

| **Communication:** | TCP/IP 100 Mbit; IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP |
| **Programming:** | Polyscope graphical user interface on 12 inch touchscreen with mounting |
| **Noise:** | Comparatively noiseless |
| **IP classification:** | IP54 |
| **Power consumption:** | Approx. 200 watts using a typical program |
| **Collaboration operation:** | 15 Advanced Safety Functions Tested in accordance with: EN ISO 13849:2008 PL d EN ISO 10218:1:2011, Clause 5.4.3 |

| **Materials:** | Aluminum, PP plastic |
| **Temperature:** | The robot can work in a temperature range of 0-50°C |
| **Power supply:** | 100-240 VAC, 50-60 Hz |
| **Cabling:** | Cable between robot and control box (6 m / 236 in) Cable between touchscreen and control box (4.5 m / 177 in) |

---

Universal Robots A/S  
Energivej 25  
DK-5230 Odense S  
Denmark  
+45 89 93 89 89  
[www.universal-robots.com](http://www.universal-robots.com)  
[sales@universal-robots.com](mailto:sales@universal-robots.com)
FT-150 Specsheet

**SIGNAL SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measuring range</td>
<td>±150 N</td>
</tr>
<tr>
<td>Effective resolution</td>
<td>±15 N·m</td>
</tr>
<tr>
<td>Signal noise</td>
<td>0.2 N</td>
</tr>
<tr>
<td>External noise sensitivity</td>
<td>Immune</td>
</tr>
<tr>
<td>Cross-talking</td>
<td>None</td>
</tr>
<tr>
<td>Drift</td>
<td>±3 N over days</td>
</tr>
<tr>
<td>Data output rate</td>
<td>100 Hz</td>
</tr>
<tr>
<td>Temperature compensation</td>
<td>15°C - 35°C</td>
</tr>
</tbody>
</table>

**MECHANICAL SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outside diameter</td>
<td>120 mm</td>
</tr>
<tr>
<td>Through-hole diameter</td>
<td>45 mm</td>
</tr>
<tr>
<td>Thickness</td>
<td>37.5 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>650 g</td>
</tr>
<tr>
<td>IP rating</td>
<td>IP54</td>
</tr>
<tr>
<td>Stiffness (calculated)</td>
<td></td>
</tr>
<tr>
<td>Fx</td>
<td>3.2 x 10^9 N/m</td>
</tr>
<tr>
<td>Fy</td>
<td>3.9 x 10^9 N/m</td>
</tr>
<tr>
<td>Mx, My, Mz</td>
<td>4700 N·m/rad</td>
</tr>
<tr>
<td>Mechanical overload</td>
<td>5x max measurement</td>
</tr>
</tbody>
</table>

**ELECTRICAL SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input voltage</td>
<td>6-28 VDC</td>
</tr>
<tr>
<td>Max power consumption</td>
<td>2 W</td>
</tr>
<tr>
<td>Sensor electrical interface</td>
<td>RS-485, RS-232, USB</td>
</tr>
</tbody>
</table>

Updated on September 10, 2014
Specifications subject to change without notice
### Data from Delay Experiments – to Moving

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Start position</th>
<th>Goal position</th>
<th>Acceleration</th>
<th>Velocity</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>-0.7</td>
<td>-3.14</td>
<td>-0.3</td>
<td>-3.14</td>
<td>1261</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.4</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.3</td>
<td>-0.7</td>
<td>0.1</td>
<td>0.3</td>
<td>1316</td>
</tr>
<tr>
<td>y</td>
<td>0.6</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.7</td>
<td>-3.14</td>
<td>-0.3</td>
<td>-3.14</td>
<td>1452</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.3</td>
<td>-0.7</td>
<td>0.1</td>
<td>0.3</td>
<td>1419</td>
</tr>
<tr>
<td>y</td>
<td>0.6</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.7</td>
<td>-3.14</td>
<td>-0.3</td>
<td>-3.14</td>
<td>1394</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.3</td>
<td>-0.7</td>
<td>0.1</td>
<td>0.3</td>
<td>1381</td>
</tr>
<tr>
<td>y</td>
<td>0.6</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.7</td>
<td>-3.14</td>
<td>-0.3</td>
<td>-3.14</td>
<td>1342</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.3</td>
<td>-0.7</td>
<td>0.1</td>
<td>0.3</td>
<td>1420</td>
</tr>
<tr>
<td>y</td>
<td>0.6</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
<td>0.3</td>
<td>1333</td>
</tr>
<tr>
<td>Measurement</td>
<td>Start position</td>
<td>Goal position</td>
<td>Acceleration</td>
<td>Velocity</td>
<td>Time(ms)</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>---------------</td>
<td>--------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>-3,14</td>
<td>-3,14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0</td>
<td>0,6</td>
<td>0,3</td>
<td>1517</td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,6</td>
<td>0</td>
<td>0,3</td>
<td>1019</td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>-3,14</td>
<td>0,1</td>
<td>-3,14</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0</td>
<td>0,6</td>
<td>0,3</td>
<td>1433</td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,6</td>
<td>0,6</td>
<td>0,3</td>
<td>1032</td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>-3,14</td>
<td>0,1</td>
<td>-3,14</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0</td>
<td>0,6</td>
<td>0,3</td>
<td>1430</td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,6</td>
<td>0,6</td>
<td>0,3</td>
<td>966</td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>-3,14</td>
<td>0,1</td>
<td>-3,14</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0</td>
<td>0,6</td>
<td>0,3</td>
<td>1404</td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,6</td>
<td>0,6</td>
<td>0,3</td>
<td>953</td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>-3,14</td>
<td>0,1</td>
<td>-3,14</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0</td>
<td>0,6</td>
<td>0,3</td>
<td>1456</td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,6</td>
<td>0,6</td>
<td>0,3</td>
<td>1057</td>
</tr>
<tr>
<td>Measurement</td>
<td>Start position</td>
<td>Goal position</td>
<td>Acceleration</td>
<td>Velocity</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>---------------</td>
<td>--------------</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>-3.14</td>
<td>0.1</td>
<td>-3.14</td>
<td>1814</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>1627</td>
</tr>
<tr>
<td>z</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0.3</td>
<td>1752</td>
</tr>
<tr>
<td>x</td>
<td>0.1</td>
<td>0.5</td>
<td>0.6</td>
<td>0.3</td>
<td>1659</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0.6</td>
<td>0.1</td>
<td>0.3</td>
<td>1766</td>
</tr>
<tr>
<td>z</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0.3</td>
<td>1683</td>
</tr>
<tr>
<td>x</td>
<td>0.1</td>
<td>0.5</td>
<td>0.6</td>
<td>0.3</td>
<td>1723</td>
</tr>
<tr>
<td>y</td>
<td>0.6</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>1663</td>
</tr>
<tr>
<td>z</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0.3</td>
<td>1828</td>
</tr>
<tr>
<td>x</td>
<td>0.1</td>
<td>0.5</td>
<td>0.6</td>
<td>0.3</td>
<td>1721</td>
</tr>
<tr>
<td>y</td>
<td>0.6</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>1814</td>
</tr>
<tr>
<td>z</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0.3</td>
<td>1627</td>
</tr>
<tr>
<td>Measurement</td>
<td>Start position</td>
<td>Goal position</td>
<td>Acceleration</td>
<td>Velocity</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>---------------</td>
<td>--------------</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>-3.14</td>
<td>0.1</td>
<td>-3.14</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>0.1</td>
<td>-3.14</td>
<td>0.1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>0.6</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>-3.14</td>
<td>0.1</td>
<td>-3.14</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>-3.14</td>
<td>0.1</td>
<td>-3.14</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>0.1</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Data from Delay Experiment – to Stationary

<table>
<thead>
<tr>
<th></th>
<th>Startpos</th>
<th>Goalpos</th>
<th>Velocity</th>
<th>TCP velocity actual</th>
<th>Deceleration</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>-0.002</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0.4</td>
<td>-0.2</td>
<td></td>
<td></td>
<td>-0.303</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>-0.001</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>-0.001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>-0.2</td>
<td>0.4</td>
<td></td>
<td></td>
<td>0.301</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>-0.003</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0.4</td>
<td>-0.2</td>
<td></td>
<td></td>
<td>-0.305</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>-0.001</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>-0.2</td>
<td>0.4</td>
<td></td>
<td></td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>0.001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0.4</td>
<td>-0.2</td>
<td></td>
<td></td>
<td>-0.296</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>-0.002</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>0.002</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>-0.2</td>
<td>0.4</td>
<td></td>
<td></td>
<td>0.304</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>-0.003</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0.4</td>
<td>-0.2</td>
<td></td>
<td></td>
<td>-0.299</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>0.001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>-0.2</td>
<td>0.4</td>
<td></td>
<td></td>
<td>0.296</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>-0.001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0.4</td>
<td>-0.2</td>
<td></td>
<td></td>
<td>-0.298</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0.3</td>
<td>10</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>-0.2</td>
<td>0.4</td>
<td></td>
<td></td>
<td>0.299</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Startpos</td>
<td>Goalpos</td>
<td>Velocity</td>
<td>TCP velocity actual</td>
<td>Deceleration</td>
<td>Time</td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>---------------------</td>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0,301</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>-0,299</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0,001</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0,304</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>-0,299</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0,3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>-0,299</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>0,001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0,302</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>-0,3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>-0,5</td>
<td>-0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,6</td>
<td>0,1</td>
<td></td>
<td></td>
<td>0,3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>Startpos</td>
<td>Goalpos</td>
<td>Velocity</td>
<td>TCP velocity actual</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>0,6</td>
<td>0,1</td>
<td>0,3</td>
<td>0,6</td>
<td>0,5</td>
<td>0,3</td>
<td>-0,184</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0,6</td>
<td>0,6</td>
<td>0,3</td>
<td>0,185</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0,6</td>
<td>0,5</td>
<td>0,3</td>
<td>0,183</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0,6</td>
<td>0,5</td>
<td>0,3</td>
<td>0,184</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0,6</td>
<td>0,5</td>
<td>0,3</td>
<td>0,184</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0,6</td>
<td>0,5</td>
<td>0,3</td>
<td>0,184</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0,6</td>
<td>0,5</td>
<td>0,3</td>
<td>0,184</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0,6</td>
<td>0,5</td>
<td>0,3</td>
<td>0,184</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0,6</td>
<td>0,5</td>
<td>0,3</td>
<td>0,184</td>
</tr>
<tr>
<td></td>
<td>Startpos</td>
<td>Goalpos</td>
<td>Velocity</td>
<td>TCP velocity actual</td>
<td>Deceleration</td>
<td>Time</td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>---------------------</td>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>0</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>-0,4</td>
<td>0,5</td>
<td></td>
<td></td>
<td>0,299</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0,001</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,004</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>0,5</td>
<td>-0,4</td>
<td></td>
<td></td>
<td>-0,294</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0,003</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>0</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>-0,4</td>
<td>0,5</td>
<td></td>
<td></td>
<td>0,296</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>0,5</td>
<td>-0,4</td>
<td></td>
<td></td>
<td>-0,298</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0,001</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>-0,4</td>
<td>0,5</td>
<td></td>
<td></td>
<td>0,297</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0,001</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>0</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>0,5</td>
<td>-0,4</td>
<td></td>
<td></td>
<td>-0,302</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>-0,001</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>0</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>-0,4</td>
<td>0,5</td>
<td></td>
<td></td>
<td>0,302</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0,002</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>0</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>0,5</td>
<td>-0,4</td>
<td></td>
<td></td>
<td>-0,3</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0,002</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,001</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>-0,4</td>
<td>0,5</td>
<td></td>
<td></td>
<td>0,3</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0,002</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0,5</td>
<td>0,5</td>
<td>0,3</td>
<td>10</td>
<td>-0,002</td>
<td>1000000</td>
</tr>
<tr>
<td>y</td>
<td>0,5</td>
<td>-0,4</td>
<td></td>
<td></td>
<td>-0,297</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Startpos</td>
<td>Goalpos</td>
<td>Velocity</td>
<td>TCP velocity actual</td>
<td>Deceleration</td>
<td>Time</td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>--------------------</td>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>-0.4</td>
<td>0.5</td>
<td>1</td>
<td>1.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0.5</td>
<td>-0.4</td>
<td>1.002</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>-0.4</td>
<td>0.5</td>
<td>1.01</td>
<td>0.008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>0.008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0.5</td>
<td>-0.4</td>
<td>0.999</td>
<td>0.004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>0.004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>-0.4</td>
<td>0.5</td>
<td>0.994</td>
<td>-0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>-0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0.5</td>
<td>-0.4</td>
<td>-0.999</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>-0.4</td>
<td>0.5</td>
<td>1.002</td>
<td>0.004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>0.004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0.5</td>
<td>-0.4</td>
<td>-1.002</td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>Startpos</td>
<td>Goalpos</td>
<td>Velocity</td>
<td>TCP velocity actual</td>
<td>Deceleration</td>
<td>Time</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>--------------------</td>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.75</td>
<td>10</td>
<td>0.004</td>
<td>1000000</td>
<td>1668</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.75</td>
<td>10</td>
<td>-0.001</td>
<td>1000000</td>
<td>700</td>
</tr>
<tr>
<td>0.6</td>
<td>0.6</td>
<td>0.75</td>
<td>10</td>
<td>0.003</td>
<td>1000000</td>
<td>1717</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.762</td>
<td>10</td>
<td>0.004</td>
<td>1000000</td>
<td>1743</td>
</tr>
<tr>
<td>0.6</td>
<td>0.6</td>
<td>-0.748</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.75</td>
<td>10</td>
<td>0</td>
<td>1000000</td>
<td>1664</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>0.1</td>
<td>0.75</td>
<td>10</td>
<td>0.004</td>
<td>1000000</td>
<td>711</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.747</td>
<td>10</td>
<td>-0.001</td>
<td>1000000</td>
<td>740</td>
</tr>
<tr>
<td>0.6</td>
<td>0.1</td>
<td>-0.751</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.749</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.75</td>
<td>10</td>
<td>-0.001</td>
<td>1000000</td>
<td>719</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>0.1</td>
<td>-0.753</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Source Code

Models

Force.cs:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace URehabAlphaBeta.Models
{
    public class Force
    {
        private Vector3D _forceVector;

        public Vector3D ForceVector
        {
            get { return _forceVector; }
            set { _forceVector = value; }
        }

        public Double TotalForce
        {
            get { return _forceVector.Length; }
            private set { } }

        public Double FX
        {
            get; 
            private set { } }

        public Double FY
        {
            get; 
            private set { } }

        public Double FZ
        {
            get; 
            private set { } }

        public Double MX
        {
            get; 
            private set { } }

        public Double MY
        {
            get; 
            private set { } }

        public Double MZ
        {
            get; 
            private set { } }

        //inverting the Z value to correspond to the coordinate system of the robot
        public Force(double fx, double fy, double fz, double mx, double my, double mz)
        {
            FX = -fx;
            FY = -fy;
            FZ = -fz;
            MX = mx;
            MY = my;
            MZ = mz;

            ForceVector = new Vector3D(FX, FY, FZ);
        }

        public Force(int[] forceData)
        {
            FX = -forceData[1];
            FY = -forceData[0];
            FZ = -forceData[2];
            MX = forceData[3];
            MY = forceData[4];
            MZ = forceData[5];

            ForceVector = new Vector3D(FX, FY, FZ);
        }

        //Calculates the magnitude of the force vector in the direction of the vector supplied in the parameter
        public Double ForceInDirection(Vector3D directionVector)
        {
            directionVector.Normalize();
            return Vector3D.DotProduct(directionVector, this.ForceVector);
        }
    }
}
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Timers;
using URehabAlphaBeta.Utilities;

namespace URehabAlphaBeta.Models
{
    public class RobotData
    {
        private Point position;
        private Point force;
        private Point tcpVelocity;
        private List<Point> tangentList;
        private Vector3D previousTangent;
        private Vector3D positionTangent;
        private Boolean tangentSet;
        private URMove lastCommand;

        public Point Position { get { return position; } private set { } }
        public Point Force { get { return force; } private set { } }
        public Point TcpVelocity { get { return tcpVelocity; } private set { } }
        public Double TcpSpeed { get { return new Vector3D(tcpVelocity.X, tcpVelocity.Y, tcpVelocity.Z).Length; } private set { } }
        public URMove LastCommand { get { return lastCommand; } set { lastCommand = value; tangentSet = false; } }
        public Vector3D PositionTangent { get { return positionTangent; } private set { positionTangent = value; PositionTangent.Normalize(); } }

        public RobotData()
        {
            position = null;
            force = null;
            tcpVelocity = null;
            tangentSet = false;
            tangentList = new List<Point>();
        }

        public void SetData(byte[] data)
        {
            Point[] points = new Point[3];
            double[] pointValues = new double[6];

            byte[] element = new byte[8];
            for (int p = 0; p < 3; p++)
            {
                int x = p * 48;
                for (int j = 0; j < 6; j++)
                {
                    for (int i = 0; i < 8; i++)
                    {
                        element[7 - i] = data[444 + (j * 8) + i + x];
                    }
                    pointValues[j] = BitConverter.ToDouble(element, 0);
                }
                points[p] = new Point(pointValues);
            }
            position = points[0];
            tcpVelocity = points[1];
            force = points[2];
            buildTangent();
        }
    }
}
private void buildTangent()
{
    if (tangentList.Count >= 3)
    {
        tangentList.Add(position);
        if (tangentList.Count > 4)
            tangentList.RemoveAt(0);
        double magnitude = Math.Abs((Vector3D.Subtract(tangentList.Last().ToVector3D(), tangentList.First().ToVector3D())).Length);
        if (magnitude > 0.0002)
        {
            PositionTangent = Vector3D.Subtract(tangentList.Last().ToVector3D(), tangentList.First().ToVector3D());
            previousTangent = PositionTangent;
            tangentSet = true;
        }
        else if (tangentSet)
            PositionTangent = previousTangent;
        else if (lastCommand != null)
        {
            PositionTangent = lastCommand.DirectionVector;
            previousTangent = positionTangent;
            tangentSet = true;
        }
        else
        {
            PositionTangent = new Vector3D(0, 0, 0);
        }
    }
    else
    {
        tangentList.Add(position);
        if (lastCommand != null)
            PositionTangent = lastCommand.DirectionVector;
        else
        {
            PositionTangent = new Vector3D(0, 0, 0);
        }
    }
}

Point.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace URehabAlphaBeta.Models
{
    public class Point
    {
        private double[] point = new double[6];
        public double X { get { return point[0]; } set { point[0] = value; } }
        public double Y { get { return point[1]; } set { point[1] = value; } }
        public double Z { get { return point[2]; } set { point[2] = value; } }
        public double RX { get { return point[3]; } set { point[3] = value; } }
        public double RY { get { return point[4]; } set { point[4] = value; } }
        public double RZ { get { return point[5]; } set { point[5] = value; } }
    }
}
public Point(double[] point)
{
    for (int i = 0; i < point.Length; i++)
    {
        this.point[i] = point[i];
    }
}

public Point(Point p)
{
    point[0] = p.X;
    point[3] = p.RX;
}

public Point(double x, double y, double z, double rx, double ry, double rz)
{
    point[0] = x;
    point[1] = y;
    point[2] = z;
    point[3] = rx;
    point[4] = ry;
    point[5] = rz;
}

public Point()
{
    X = 0;
    Y = 0;
    Z = 0;
    RX = 0;
    RY = 0;
    RZ = 0;
}

public double[] getPoint()
{
    return this.point;
}

public Vector3D ToVector3D()
{
    return new Vector3D(X, Y, Z);
}

public override string ToString()
{
    return String.Format("X:{0} Y:{1} Z:{2} RX:{3} RY:{4} RZ:{5}", this.X, this.Y, this.Z, this.RX, this.RY, this.RZ);
}


URMove.cs:

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using URehabAlphaBeta.Models;
namespace URehabAlphaBeta.Utilities
{
    public class URMove
    {
        protected String moveType;
        protected Point point;
        protected Point midPoint;
        protected double velocity;
        protected double acceleration;
        protected Vector3D directionVector;

        public static String Stop { get { return "stop(a=10)\n"; } private set {} }

        public String MoveType { get { return moveType; } set { moveType = value; } }
        public Vector3D DirectionVector { get { return directionVector; } set { directionVector = value; } }
        public Point Point { get { return point; } set { point = value; } }
        public Point MidPoint { get { return midPoint; } set { midPoint = value; } }
        public Double Velocity { get { return velocity; } set { velocity = value; } }

        public URMove()
        {
        }

        public URMove(URMove move, double speed)
        {
            this.point = new Point(move.point);
            this.midPoint = new Point(move.midPoint);
            this.moveType = move.moveType;
            this.directionVector = move.directionVector;
            this.velocity = speed;
        }

        public URMove(Point point)
        {
            this.point = point;
            this.midPoint = new Point(0,0,0,0,0,0);
            this.velocity = 0.2;
        }

        public URMove(Point point, String moveType)
        {
            this.point = point;
            this.midPoint = new Point(0, 0, 0, 0, 0, 0);
            this.velocity = 0.2;
            this.moveType = moveType;
        }

        public URMove(Point point, Point midPoint)
        {
            this.point = point;
            this.midPoint = midPoint;
            this.velocity = 0.2;
        }

        public URMove(Point point, Point midPoint, String moveType)
        {
            this.point = point;
            this.midPoint = midPoint;
            this.moveType = moveType;
            this.velocity = 0.2;
        }

        public URMove(Point point, Point midPoint, double velocity)
        {
            this.point = point;
            this.midPoint = midPoint;
            this.velocity = velocity;
        }

        public URMove(Point point, double velocity)
        {
            this.point = point;
            this.midPoint = new Point(0, 0, 0, 0, 0);
            this.velocity = velocity;
        }
    }
}
```csharp
public void AddDestinationPoint(Point point)
{
    this.point = point;
}

public void ChangeSpeed(double velocity)
{
    this.velocity = velocity;
}

public override string ToString()
{
    double d = 0;
    double s = 3.14;

    if (moveType.Equals("movec"))
    {
        return moveType + "(" +
            midPoint.X.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", ", " +
            midPoint.Y.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            midPoint.Z.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            d.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            s.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            d.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            "], a=0.25"
        + ", v=" + velocity.ToString(CultureInfo.GetCultureInfo("en-GB")) + ")\n";
    }
    else
    {
        return moveType + "(" +
            point.X.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            point.Y.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            point.Z.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            d.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            s.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " +
            d.ToString(CultureInfo.GetCultureInfo("en-GB")) + "]" + ", a=0.25" + ", v=" + velocity.ToString(CultureInfo.GetCultureInfo("en-GB")) + ")\n";
    }
}

public class URMoveC : URMove
{
    public URMoveC()
    {
        this.moveType = "movec";
    }

    public URMoveC(Point midPoint)
    {
        this.moveType = "movec";
        this.midPoint = midPoint;
    }

    public URMoveC(Point point, Point midPoint, double velocity) : base(point, midPoint, velocity)
    {
        this.moveType = "movec";
    }

    public override string ToString()
    {
        double d = 0;
        double s = 3.14;
```
return moveType + "(p[ + 
midPoint.X.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
midPoint.Y.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
midPoint.Z.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
d.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
s.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
d.ToString(CultureInfo.GetCultureInfo("en-GB")) + "]], a=0.15" + ", v=" + velocity.ToString(CultureInfo.GetCultureInfo("en-GB")) + ")"\n};

public class URMoveL : URMove
{
    public URMoveL(Point point) : base(point)
    {
        this.moveType = "movel";
    }
}

public class URMoveP : URMove
{
    public URMoveP(Point point) : base(point)
    {
        this.moveType = "movep";
    }
    public URMoveP()
    {
        this.moveType = "movep";
    }
    public URMoveP(Point point, double velocity) : base(point, velocity)
    {
        this.moveType = "movep";
    }
    public override string ToString()
    {
        double d = 0;
        double s = 3.14;
        return moveType + "(p[ + 
point.X.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
point.Y.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
point.Z.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
d.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
s.ToString(CultureInfo.GetCultureInfo("en-GB")) + ", " + 
d.ToString(CultureInfo.GetCultureInfo("en-GB")) + "]]", a=0.15" + ", v=" + velocity.ToString(CultureInfo.GetCultureInfo("en-GB")) + ")\n};

public class URMoveJ : URMove
{
    public URMoveJ(Point point) : base(point)
    {
        this.moveType = "movej";
    }
}
Exercise.cs:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using URehabAlphaBeta;
using URehabAlphaBeta.Utilities;

namespace URehabAlphaBeta.Models
{
    public class Exercise
    {
        private String name;
        private URMove[] route;
        private Point[] middlePoint;
        private int[] waitTime;

        public int[] WaitTime { get { return waitTime; } set { waitTime = value; } }
        public URMove[] Route { get { return route; } set { route = value; } }
        public String Name { get { return name; } set { name = value; } }

        public Exercise (URMove[] points, int[] waitTime, String name)
        {
            route = points;
            this.waitTime = waitTime;
            this.name = name;
        }

        public Exercise()
        {
        }
    }
}
```

Robot.cs:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using URehabAlphaBeta.Utilities;
using System.Windows;
using URehabAlphaBeta.Models;

namespace URehabAlphaBeta
{
    public delegate void ForceDataDelegate(ForceEventArgs e);

    public static class Robot
    {
        public delegate void SenderResponseDelegate(SenderResponseEventArgs _args);
        public delegate void ConnectionDelegate(EventArgs e);

        public static event SenderResponseDelegate OnSenderResponse;
        public static event ConnectionDelegate OnDataReceived;
        public static event ConnectionDelegate OnConnected;
        public static event ForceDataDelegate ForceDataReceived;
        private static RobotData currentData = new RobotData();
        private static Client robotDataReceiver;
        private static Client robotCommander;

        public static RobotData CurrentData { get { return currentData; } private set { } }
```
public static Boolean SenderConnected { get { if (robotCommander == null) return false; return robotCommander.IsConnected; } private set {} }

public static Boolean ReceiverConnected { get { if (robotDataReceiver == null) return false; return robotDataReceiver.IsConnected; } private set {} }

//Connect to the 125hz data port on the robot
public static void ConnectReceiver()
{
    if (robotDataReceiver == null)
    {
        robotDataReceiver = new Client();
        robotDataReceiver.OnDataRecievedEvent += Receiver_OnDataReceived;
        robotDataReceiver.OnConnectedEvent += Receiver_OnConnectedEvent;
        robotDataReceiver.StartClient("192.168.1.18", 30003, true);
    }
}

//Connect to the Force Sensor
public static Boolean ConnectForceSensor()
{
    return ForceDataRead.Connect();
}

public static void DisconnectForceSensor()
{
    ForceDataRead.Disconnect();
}

//Triggers OnConnected event when connected to the data port on the robot
private static void Receiver_OnConnectedEvent(EventArgs _args)
{
    if (OnConnected != null)
        OnConnected(EventArgs.Empty);
}

//Connect to the primary control port on the robot
public static void ConnectSender()
{
    if (robotCommander == null)
    {
        robotCommander = new Client();
        robotCommander.OnSenderResponse += Sender_OnSenderResponse;
        robotCommander.StartClient("192.168.1.18", 30001, false);
    }
}

//Triggers event when receiving response from the control port
private static void Sender_OnSenderResponse(SenderResponseEventArgs _args)
{
    if (OnSenderResponse != null)
        OnSenderResponse(_args);
}

//Send a command to the control port on the robot
public static void SendCommand(String command)
{
    robotCommander.SendCommand(command);
}

//Send a URMove command to the control port on the robot
public static void SendCommand(URMove command)
{
    robotCommander.SendCommand(command.ToString());
    CurrentData.LastCommand = command;
}

//Close the data port connection
public static void StopReceiver()
{
    if (robotDataReceiver != null)
    {
        robotDataReceiver.CloseClient();
        robotDataReceiver = null;
    }
}
} //Close the control port connection
public static void StopSender()
{
    if (robotCommander != null)
    {
        robotCommander.CloseClient();
        robotCommander = null;
    }
}

//Triggers event when receiving data from data port and setting the data to the CurrentData property
private static void Receiver_OnDataReceived(DataRecievedEventArgs _args)
{
    currentData.SetData(_args.Data);
    if (OnDataReceieved != null)
        OnDataReceieved(EventArgs.Empty);
}

SenderResponseEventArgs.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace URehabAlphaBeta
{
    public class SenderResponseEventArgs : EventArgs
    {
        private String _response;
        public SenderResponseEventArgs(String response)
        {
            _response = response;
        }
        public String Response { get { return _response; } }
    }
}

DataReceivedEventArgs.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace URehabAlphaBeta
{
    public class DataReceivedEventArgs : EventArgs
    {
        private byte[] data;
        public DataReceivedEventArgs(byte[] data)
        {
            this.data = data;
        }
    }
}

154
public Byte[] Data { get { return data; } }

Client.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace URehabAlphaBeta
{
    public class StateObject
    {
        // Client socket.
        public Socket workSocket = null;
        // Size of receive buffer.
        public const int BufferSize = 1044;
        // Receive buffer.
        public byte[] buffer = new byte[BufferSize];
        // Received data string.
        public StringBuilder sb = new StringBuilder();
    }

    public class Client
    {
        // The port number for the remote device.
        //private const int port = 30002;
        // ManualResetEvent instances signal completion.
        private static ManualResetEvent connectDone =
            new ManualResetEvent(false);
        private static ManualResetEvent sendDone =
            new ManualResetEvent(false);
        private static ManualResetEvent receiveDone =
            new ManualResetEvent(false);
        private static ManualResetEvent receiveAborted =
            new ManualResetEvent(false);
        // The response from the remote device.
        public string response = String.Empty;
        public byte[] data = new byte[1044];
        public bool IsConnected { get { return client.Connected; } private set { } }
        private Socket client;
        private bool abortReceive;
        public delegate void DataRecievedDelegate(DataReceivedEventArgs _args);
        public delegate void SenderResponseDelegate(SenderResponseEventArgs _args);
        public delegate void ConnectionDelegate(EventArgs _args);
        public event DataRecievedDelegate OnDataRecievedEvent;
        public event SenderResponseDelegate OnSenderResponse;
        public event ConnectionDelegate OnConnectedEvent;
        public void StartClient(string ip, int port, bool receiver) { }
}
// Connect to a remote device.
try
{
    // Establish the remote endpoint for the socket.
    IPAddress ipAddress = IPAddress.Parse(ip);
    IPEndPoint remoteEP = new IPEndPoint(ipAddress, port);
    // Create a TCP/IP socket.
    client = new Socket(AddressFamily.InterNetwork,
                         SocketType.Stream, ProtocolType.Tcp);
    // Connect to the remote endpoint.
    client.BeginConnect(remoteEP,
                        new AsyncCallback(ConnectCallback), client);
    connectDone.WaitOne(100);
    if (receiver)
        Receive();
}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
}

public void CloseClient()
{
    try
    {
        abortReceive = true;
        receiveAborted.WaitOne();
        connectDone.Reset();
        // Release the socket.
        client.Shutdown(SocketShutdown.Both);
        client.Close();
    }
    catch (Exception e)
    {
        Console.WriteLine("Error: " + e.ToString());
    }

    public void SendCommand(String command)
    {
        Send(command);
        sendDone.WaitOne();
    }

    private void ConnectCallback(IAsyncResult ar)
    {
        try
        {
            // Retrieve the socket from the state object.
            Socket client = (Socket)ar.AsyncState;
            // Complete the connection.
            client.EndConnect(ar);
            if (OnConnectedEvent != null)
                OnConnectedEvent(EventArgs.Empty);
            Console.WriteLine("Socket connected to {0}",
                              client.RemoteEndPoint.ToString());
            // Signal that the connection has been made.
            connectDone.Set();
        }
        catch (Exception e)
        {
            Console.WriteLine(e.ToString());
        }
    }

    public void ReceiveData()
private void Receive()
{
    try
    {
        // Create the state object.
        StateObject state = new StateObject();
        state.workSocket = client;

        // Begin receiving the data from the remote device.
        client.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
                             new AsyncCallback(ReceiveCallback), state);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
}

private void SendReceive()
{
    try
    {
        // Create the state object.
        StateObject state = new StateObject();
        state.workSocket = client;

        // Begin receiving the data from the remote device.
        client.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
                             new AsyncCallback(ReceiveSendCallback), state);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
}

private void ReceiveCallback(IAsyncResult ar)
{
    try
    {
        // Retrieve the state object and the client socket
        // from the asynchronous state object.
        StateObject state = (StateObject)ar.AsyncState;
        Socket client = state.workSocket;

        // Read data from the remote device.
        int bytesRead = client.EndReceive(ar);
        if (bytesRead > 0)
        {
            // There might be more data, so store the data received so far.
            response = Encoding.ASCII.GetString(state.buffer, 0, bytesRead);
            if (OnDataRecievedEvent != null)
            {
                OnDataRecievedEvent(new DataReceivedEventArgs(state.buffer));
            }
            if (!abortReceive)
            {
                // get the rest of the data.
                client.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
                                     new AsyncCallback(ReceiveCallback), state);
            }
            else
            {
                receiveAborted.Set();
            }
        }
        else
        {
            // All the data has arrived; put it in response.
        }
    }
}
if (state.sb.Length > 1)
{
    response = state.sb.ToString();
}
    // Signal that all bytes have been received.
}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
}

private void ReceiveSendCallback(IAsyncResult ar)
{
    try
    {
        // Retrieve the state object and the client socket
        // from the asynchronous state object.
        StateObject state = (StateObject)ar.AsyncState;
        Socket client = state.workSocket;
        // Read data from the remote device.
        int bytesRead = client.EndReceive(ar);
        response = (Encoding.ASCII.GetString(state.buffer, 0, bytesRead));
        receiveDone.Set();
        if (OnSenderResponse != null)
            OnSenderResponse(new SenderResponseEventArgs(response));
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
}

private void Send(String data)
{
    // Convert the string data to byte data using ASCII encoding.
    byte[] byteData = Encoding.ASCII.GetBytes(data);
    // Begin sending the data to the remote device.
    client.BeginSend(byteData, 0, byteData.Length, 0,
        new AsyncCallback(SendCallback), client);
}

private void SendCallback(IAsyncResult ar)
{
    try
    {
        // Retrieve the socket from the state object.
        Socket client = (Socket)ar.AsyncState;
        // Complete sending the data to the remote device.
        int bytesSent = client.EndSend(ar);
        Console.WriteLine("Sent {0} bytes to server.", bytesSent);
        // Signal that all bytes have been sent.
        sendDone.Set();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
}
using System;
using System.Collections.Generic;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;
using URehabAlphaBeta.Models;
namespace URehabAlphaBeta.Utilities
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public static class ForceDataRead
    {
        private static SerialPort port = new SerialPort("COM3", 19200, Parity.None, 8, StopBits.One);
        private static int[] _data = new int[6];
        private static int[] _offset = new int[6] { 0, 0, 0, 0, 0, 0 };
        private static bool _forceConnected;
        private static List<int[]> _dataList = new List<int[]>();
        public static int[] ForceData {
            get { return getNormalisedForce(); } 
            private set {
                _data = value;
                if (newData != null)
                    newData(new ForceDataEventArgs(new Force(ForceData)));
            }
        }
        public static bool ForceConnected {
            get { return _forceConnected; } 
            private set { }
        }
        public static event ForceDataDelegate newData;
        public static bool Connect()
        { 
            port.DataReceived += new SerialDataReceivedEventHandler(DataReceivedHandler);
            try
            {
                port.Open();
                //Send starting sequence for receiving data
                byte[] startReadCommand = new byte[8] { 0x09, 0x06, 0x01, 0x9A, 0x02, 0x00, 0x31, 0x58
                };
                byte[] startReadCommand2 = new byte[8] { 0x09, 0x03, 0x01, 0xFE, 0x00, 0x05, 0xE4, 0x8D
                };
                byte[] startReadCommand3 = new byte[8] { 0x09, 0x03, 0x01, 0xFE, 0x00, 0x05, 0xE4, 0x8D
                };
                byte[] startReadCommand4 = new byte[8] { 0x09, 0x03, 0x01, 0xF4, 0x00, 0x03, 0x44, 0x8D
                };
                byte[] startReadCommand5 = new byte[11] { 0x09, 0x10, 0x01, 0x9A, 0x00, 0x01, 0x02, 0x02, 0x00 CD, 0xCA };
                //09 10 01 9A 00 01 02 00 CD CA          09 03 01 F4 00 03 44 8D
                port.Write(startReadCommand, 0, 8);
                Thread.Sleep(100);
                port.Write(startReadCommand2, 0, 8);
            }
        }
    }
}
Thread.Sleep(100);
port.Write(startReadCommand3, 0, 8);
Thread.Sleep(100);
port.Write(startReadCommand4, 0, 8);
Thread.Sleep(100);
port.Write(startReadCommand5, 0, 11);
Thread.Sleep(100);
_forceConnected = true;
return true;

} catch (Exception e)
{
  return false;
}

public static void Disconnect()
{
  if (port.IsOpen)
    port.Close();
}

public static void DataReceivedHandler(object sender, EventArgs e)
{
  if (port.BytesToRead >= 32)
  {
    byte[] buffer = new byte[port.BytesToRead];
    port.Read(buffer, 0, port.BytesToRead);
    int[] dataOut = new int[6];
    int startindex = 0;
    bool startFound = false;
    while (startindex <= 16 && !startFound)
    {
      if (buffer[startindex] == 0x20 && buffer[startindex + 1] == 0x4E)
        startFound = true;
      startindex++;
    }
    for (int i = 0; i < 6; i++)
    {
      dataOut[i] = BitConverter.ToInt16(buffer, startindex + 1 + (2 * i));
    }
    ForceData = dataOut;
    if (_dataList.Count < 10)
    {
      _dataList.Add(dataOut);
    }
    else
    {
      _dataList.RemoveAt(0);
      _dataList.Add(dataOut);
    }
    port.DiscardInBuffer();
  }
}

public static void ForceToZero()
{
  _offset = _data;
  if (_dataList.Count != 0) {
    int[] average = new int[6];
    for (int i = 0; i < _dataList.Count; i++)
    {
      for (int k = 0; k < 6; k++)
        average[k] += _dataList[i][k];
    }
    for (int i = 0; i < 6; i++)
      average[i] /= _dataList.Count;
  _offset = average;
private static int[] getNormalisedForce()
{
    int[] normalisedForce = new int[6];
    for (int i = 0; i < 6; i++)
    {
        normalisedForce[i] = _data[i] - _offset[i];
    }
    return normalisedForce;
}

namespace URehabAlphaBeta
{
    public class ForceDataEventArgs
    {
        public Force ForceData {
            get; private set; }
        public ForceDataEventArgs(Force forceData)
        {
            ForceData = forceData;
        }
    }
}

namespace URehabAlphaBeta
{
    public delegate void ProximityDelegate(EventArgs e);
    public class Tracker
    {
        private Point _point;
        private double trackingRadius;
        public event ProximityDelegate OnApproachingPoint;
        public Tracker()
        {
            trackingRadius = 1;
        }
        public Tracker(double trackingRadius)
this.trackingRadius = trackingRadius;
}

//Set the position to signal when approaching
public void setPointToTrack(Point _point)
{
    this._point = _point;
    Robot.OnDataReceived += DataAccessor_OnDataReceieved;
}

//Checks if approaching the position that is being tracked
private void DataAccessor_OnDataReceieved(EventArgs e)
{
    Point position = Robot.CurrentData.Position;
    Vector3D vec = new Vector3D(_point.X, _point.Y, _point.Z);
    Vector3D vec2 = new Vector3D(position.X, position.Y, position.Z);
    double length = Vector3D.Subtract(vec, vec2).Length;
    double trackR = trackingRadius * (Robot.CurrentData.TcpSpeed*0.25);
    if(length < trackR && length > -trackR)
    {
        if(OnApproachingPoint != null)
        {
            OnApproachingPoint(EventArgs.Empty);
            Robot.OnDataReceived -= DataAccessor_OnDataReceieved;
        }
    }
}

ExecuteExercise.cs:

using URehabAlphaBeta.Utilities;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Timers;
using URehabAlphaBeta.Models;
namespace URehabAlphaBeta
{
    public class ExecuteExercise
    {
        private BackgroundWorker bw = new BackgroundWorker();
        public delegate void ChangedEventHandler(object sender, EventArgs e);
        public event ChangedEventHandler StatusChanged;
        public event ChangedEventHandler PercentChanged;
        public event ChangedEventHandler RepetitionsChanged;
        private int _status;
        private int _percent = 0;
        private Exercise _currEx;
        private int _iterator;
        private Tracker _tracker;
        private Timer _timer;
        private Timer _UpdateTimer;
        private Timer _pauseTimer;
        private Boolean waiting;
    }
}
private bool testRun;
private URMove move;
private int _repetitions;
private bool reverse;
private int dirvecCounter = 0;
private int resistance;

private const Double FORCE_TO_SPEED_SCALAR = 0.0001;

public int Status { get { return _status; } private set { _status = value; } }
OnStatusChanged(EventArgs.Empty); } }
public int Percent { get { return _percent; } private set { _percent = value; } }
OnPercentChanged(EventArgs.Empty); } }
public int Repetitions { get { return _repetitions; } set { _repetitions = value; } }
public int Resistance { get { return resistance; } set { resistance = value; } }

public ExecuteExercise(Exercise currEx)
{
    _currEx = currEx;
    _iterator = 0;
    waiting = false;
    Status = ExerciseStatus.NOTREADY;
    _tracker = new Tracker();
    _tracker.OnApproachingPoint += closeEnough;
    _timer = new Timer();
    _timer.Elapsed += timeIsUp;
    _UpdateTimer = new Timer(100);
    _UpdateTimer.Elapsed += _UpdateTimer_Elapsed;
    _pauseTimer = new Timer(500);
    _pauseTimer.Elapsed += _pauseTimer_Elapsed;
    _repetitions = 1;
    Resistance = 5;
    reverse = false;
}

public ExecuteExercise(Exercise currEx, bool testRun, double trackingRadius)
{
    _currEx = currEx;
    this.testRun = testRun;
    _iterator = 0;
    waiting = false;
    Status = ExerciseStatus.NOTREADY;
    _tracker = new Tracker(trackingRadius);
    _tracker.OnApproachingPoint += closeEnough;
    _timer = new Timer();
    _timer.Elapsed += timeIsUp;
    _UpdateTimer = new Timer(100);
    _UpdateTimer.Elapsed += _UpdateTimer_Elapsed;
    _pauseTimer = new Timer(500);
    _pauseTimer.Elapsed += _pauseTimer_Elapsed;
    _repetitions = 1;
    Resistance = 5;
    reverse = false;
}
// Pause between repetitions
private void _pauseTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    _pauseTimer.Stop();
    nextMove();
    _UpdateTimer.Start();
}

// Updates the speed
private void _UpdateTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    if (!waiting && !testRun)
    else if (testRun)
    {
        Vector3D moveDirVect = _currEx.Route[_iterator - 1].DirectionVector;
        if (_iterator <= _currEx.Route.Length && moveDirVect.X == 0 && moveDirVect.Y == 0 && moveDirVect.Z == 0)
        {
            if (dirvecCounter == 4)
            {
                _currEx.Route[_iterator - 1].DirectionVector = Vector3D.Subtract(nowPoint, startPoint);
                dirvecCounter = 0;
            }
            dirvecCounter++;
        }
    }
}

public Exercise GetExercise()
{
    return _currEx;
}

// Sets the robot in starting position of the exercise
public void Prepare()
{
    sendCommand(_currEx.Route[0]);
    _tracker.setPointToTrack(_currEx.Route[0].Point);
}

// Starts the exercise
public void Start()
{
    _UpdateTimer.Start();
    Status = ExerciseStatus.STARTED;
    nextMove();
}

private void timeIsUp(Object sender, EventArgs e)
{
    waiting = false;
    _timer.Stop();
    nextMove();
}

private void closeEnough(EventArgs e)
{
    if (_iterator == 0)
    {
        _iterator++;
        Status = ExerciseStatus.READY;
    }
    else
    {
        if (_currEx.getWaitTime()[_iterator - 1] == 0)
        {
            // More code here...
        }
    }
}
if (testRun && _currEx.Route[_iterator - 1].MidPoint.X != 0 && _currEx.Route[_iterator - 1].MidPoint.Y != 0 && _currEx.Route[_iterator - 1].MidPoint.Z != 0) {
} else {
    if (testRun && _currEx.Route[_iterator - 1].MidPoint.X != 0 && _currEx.Route[_iterator - 1].MidPoint.Y != 0 && _currEx.Route[_iterator - 1].MidPoint.Z != 0) {
    } else {
        _timer.Interval = _currEx.getWaitTime()[_iterator - 1];
        _timer.Start();
        waiting = true;
    }
}

//Contains the logic for selecting or creating the next move to be performed in the exercise
private void nextMove() {
    //When moving backwards from end position to starting position
    if (reverse) {
        if (_iterator > 1) {
            move = new URMove(_currEx.Route[_iterator - 1], 0.2);
            if (move.MidPoint.X == 0 && move.MidPoint.Y == 0 && move.MidPoint.Z == 0)
                sendCommand(move);
            else {
                Point p = _currEx.Route[_iterator - 2].Point;
                move.Point.X = p.X;
                move.Point.Y = p.Y;
                _iterator--;
                String moveM = move.ToString();
                sendCommand(move);
            }
            _tracker.setPointToTrack(move.Point);
        } else {
            reverse = false;
            _UpdateTimer.Stop();
            _pauseTimer.Start();
        }
    } else {
        if (_iterator < _currEx.Route.Length) {
            if (_repetitions <= 1)
                Status = ExerciseStatus.FINISHED;
            _UpdateTimer.Stop();
        } else {
            if (_repetitions <= 1)
                Status = ExerciseStatus.FINISHED;
            _UpdateTimer.Stop();
        }
    }
}

//Moving from start position to end position
else {
    if (_iterator < _currEx.Route.Length) {
        if (_repetitions <= 1)
            Status = ExerciseStatus.FINISHED;
        _UpdateTimer.Stop();
    } else {
        if (_repetitions <= 1)
            Status = ExerciseStatus.FINISHED;
        _UpdateTimer.Stop();
    }
}
_iterator = 0;
}
else {
    reverse = true;
    Repetitions--; 
    if (RepetitionsChanged != null)
        RepetitionsChanged(this, EventArgs.Empty);
    _UpdateTimer.Stop();
    _pauseTimer.Start();
}
}
else {
    move = new URMove(currEx.Route[_iterator], 0.2);
    sendCommand(move);
    _tracker.setPointToTrack(move.Point);
    _iterator++;
}
}

//Send movement command to robot
private void sendCommand(URMove moveCommand)
{
    Robot.SendCommand(moveCommand);
}

//Change the speed of the TCP based on the force parameter
private void changeSpeed(double force)
{
    double speed = force * FORCE_TO_SPEED_SCALAR * Resistance;
    if (speed > 0.3)
        speed = 0.3;
    else if (speed < 0)
        speed = 0.001;
    if (reverse)
    {
        if (_iterator > 0)
        {
            Vector3D distance = Vector3D.Subtract(Robot.CurrentData.Position.ToVector3D(),
            move.Point.ToVector3D());
            if (distance.Length - move.DirectionVector.Length < 0.02)
            {
                move = new URMove(move.Point,"movep");
                move.ChangeSpeed(speed);
                Robot.SendCommand(move);
            }
            else {
                move.ChangeSpeed(speed);
                Robot.SendCommand(move);
            }
        }
        else
        {
            _UpdateTimer.Stop();
        }
    }
    else {
        if (_iterator <= _currEx.Route.Length)
        {
            Vector3D distance = Vector3D.Subtract(Robot.CurrentData.Position.ToVector3D(),
            move.Point.ToVector3D());
            if (distance.Length - move.DirectionVector.Length < 0.03)
            {
                move = new URMove(move.Point,"movep");
                move.ChangeSpeed(speed);
                Robot.SendCommand(move);
            }
            else {
                move.ChangeSpeed(speed);
                Robot.SendCommand(move);
            }
        }
    }
}
else
    _UpdateTimer.Stop();
}

protected virtual void OnPercentChanged(EventArgs e)
{
    if (PercentChanged != null)
        PercentChanged(this, e);
}

protected virtual void OnStatusChanged(EventArgs e)
{
    if (StatusChanged != null)
        StatusChanged(this, e);
}

// Status class with the different status states
public static class ExerciseStatus
{
    public static int NOTREADY = 0;
    public static int READY = 1;
    public static int STARTED = 2;
    public static int FINISHED = -1;
}

namespace URehabAlphaBeta
{
    public class FreeMove
    {
        Timer updateTimer;
        Timer pauseTimer;

        private Vector3D previousForceVector;
        private Vector3D previousPositionVec;
        private double maxSpeed;

        private Vector3D forceV;
        private List<Force> forceList;

        const double FORCE_TO_SPEED_SCALAR = 0.0001;

        private double friction;
        private bool stopped;

        public Double MaxSpeed { get { return maxSpeed; } set { maxSpeed = value; } }

        public FreeMove()
        {
            updateTimer = new Timer(70);
            updateTimer.Elapsed += UpdateTimer_Elapsed;
            updateTimer.Start();
        }
    }
}
pauseTimer = new Timer();
pauseTimer.Elapsed += PauseTimer_Elapsed;

forceList = new List<Force>();
ForceDataRead.newData += ForceDataRead_newData;

friction = 1;
maxSpeed = 0.25;
stopped = false;
}

public FreeMove(double friction) : this()
{
    // Adjusts lower bound of force required to initiate motion
    this.friction = friction;
}

private void PauseTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    pauseTimer.Stop();
    updateTimer.Start();
}

private void ForceDataRead_newData(ForceEventArgs e)
{
    if (forceList.Count < 10)
    {
        forceList.Add(e.ForceData);
    }
    else
    {
        forceList.RemoveAt(0);
        forceList.Add(e.ForceData);
    }
    for(int i = 0; i < forceList.Count; i++)
    {
        forceV.X += forceList[i].FX;
        forceV.Y += forceList[i].FY;
        forceV.Z += forceList[i].FZ;
    }
    forceV.X /= forceList.Count;
    forceV.Y /= forceList.Count;
    forceV.Z /= forceList.Count;
}

// Stops movement for the given amount of time in milliseconds
public void Stop(int time)
{
    updateTimer.Stop();
    pauseTimer.Interval = time;
    pauseTimer.Start();
    Robot.SendCommand(URMove.Stop);
}

// Stops adjusting the movement
public void Stop()
{
    updateTimer.Stop();
    Robot.SendCommand(URMove.Stop);
}

// Starts adjusting the movement
public void Start()
{
    updateTimer.Start();
}

// Creates the next movement command
private void UpdateTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    Force force = new Force(ForceDataRead.ForceData);
    double speed = force.TotalForce * FORCE_TO_SPEED_SCALAR;
    Vector3D forceVec = force.ForceVector;
forceVec.Normalize();
forceVec.X *= 0.5;
forceVec.Y *= 0.5;
forceVec.Z *= 0.5;

String command;

Point p = Robot.CurrentData.Position;

//Creates a position vector from the current position and the force vector
Vector3D positionVec = Vector3D.Add(p.ToVector3D(), forceV);

//Controls the maximum speed allowed
if (speed > maxSpeed)
speed = maxSpeed;

//Scales new position to make sure its not too close to current position
if (positionVec.Length < 0.1)
{
    double scale = 0.1 / positionVec.Length;
    positionVec.X *= scale;
    positionVec.Y *= scale;
    positionVec.Z *= scale;
}

if (previousForceVector != null)
{
    //Creates a vector in the middle of the current and previous force vector
    Vector3D midDirectionVec = Vector3D.Add(forceV, previousForceVector);
    double scaleValue = (forceV.Length / previousForceVector.Length)/2;
    midDirectionVec.X *= scaleValue;
    midDirectionVec.Y *= scaleValue;
    midDirectionVec.Z *= scaleValue;

    //Creates a vector with a direction between the current and previous position vector
    Vector3D midPositionVec = Vector3D.Add(positionVec, previousPositionVec);
    midPositionVec.X *= 0.5;
    midPositionVec.Y *= 0.5;
    midPositionVec.Z *= 0.5;

    double stopAcceleration = Robot.CurrentData.TcpSpeed * 2;
    double startAcceleration = speed * 4;

    //FreeMove control logic
    if (stopped && Robot.CurrentData.TcpSpeed > 0.10)
    {
        command = String.Format("stopl(a=1)\n");
    }
    else if (speed < 0.005 * friction)
    {
        command = String.Format("stopl(a=0)\n", stopAcceleration.ToString(CultureInfo.GetCultureInfo("en-GB")));
        stopped = true;
    }
    else if (Vector3D.AngleBetween(previousPositionVec, forceV) > 10)
    {
        command = String.Format("movep(p[0,1,2,3,4,5], a=7, v=6)\n", midPositionVec.X.ToString(CultureInfo.GetCultureInfo("en-GB")), midPositionVec.Y.ToString(CultureInfo.GetCultureInfo("en-GB")), midPositionVec.Z.ToString(CultureInfo.GetCultureInfo("en-GB")), 0, 3.14.ToString(CultureInfo.GetCultureInfo("en-GB")), 0, speed.ToString(CultureInfo.GetCultureInfo("en-GB"))), 0,
    }
    else
    {
        command = String.Format("movep(p[0,1,2,3,4,5], a=7, v=6)\n", positionVec.X.ToString(CultureInfo.GetCultureInfo("en-GB")), positionVec.Y.ToString(CultureInfo.GetCultureInfo("en-GB")), positionVec.Z.ToString(CultureInfo.GetCultureInfo("en-GB"))), 0,
```csharp
}
else
    command = String.Format("movep[p[{0},{1},{2},{3},{4},{5}], a=0.1, v={6}]" + "\n",
positionVec.X.ToString(CultureInfo.GetCultureInfo("en-GB")),
positionVec.Y.ToString(CultureInfo.GetCultureInfo("en-GB")),
positionVec.Z.ToString(CultureInfo.GetCultureInfo("en-GB")), 0,
)
Robot.SendCommand(command);
previousForceVector = forceV;
previousPositionVec = positionVec;
}
}

ExerciseXML.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.IO;
using System.Xml.Serialization;
using URehabAlphaBeta.Models;
namespace URehabAlphaBeta.Utilities
{
    public static class ExerciseXML
    {
        //Create XML file from Exercise object
        public static Boolean ToXML(Exercise exIn)
        {
            Exercise ex = exIn;
            var writer = new XmlSerializer(typeof(Exercise));
            var wfile = new StreamWriter("Exercises/" + ex.Name + ".xml");
            writer.Serialize(wfile, ex);
            wfile.Close();
            return true;
        }

        //Create Exercise object from XML file with the name supplied in the parameter
        public static Exercise FromXML(string name)
        {
            XmlSerializer reader = new XmlSerializer(typeof(Exercise));
            StreamReader file = new StreamReader(name);
            Exercise ex = (Exercise)reader.Deserialize(file);
            file.Close();
            return ex;
        }
    }
}

View Models

___________________________________________________________________________
________
ViewModelBase.cs:

```csharp
using URehabAlphaBeta.Utilities;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using URehabAlphaBeta.Models;

namespace URehabAlphaBeta.ViewModels
{
    public class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected static MainViewModel mainViewModel;
        public static List<ViewModelBase> ViewChain = new List<ViewModelBase>();
        public static List<ViewModelBase> ViewModelList = new List<ViewModelBase>();

        public ViewModelBase()
        {
            if (!ViewModelBase.ViewModelList.Contains(this))
                ViewModelList.Add(this);
            if (ForceDataRead.ForceConnected == true)
                ForceDataRead.ForceToZero();
        }

        //Implementation of INotifyPropertyChanged interface allowing for data binding in all the ViewModel classes
        protected void OnPropertyChanged(String name)
        {
            PropertyChangedEventHandler handler = this.PropertyChanged;
            if (null != handler)
                handler(this, new PropertyChangedEventArgs(name));
        }

        //Loads the previous view model used and removes it from the ViewChain
        protected void goBack()
        {
            mainViewModel.ViewModel = ViewChain.Last();
            ViewChain.Remove(ViewChain.Last());
        }

        //Sets a new view model of the supplied type. If it already exist it is selected from the ViewModelList
        protected bool setViewModelByType(Type type)
        {
            bool found = false;
            foreach (ViewModelBase vm in ViewModelList)
            {
                if (vm.GetType().Equals(type))
                {
                    mainViewModel.ViewModel = vm;
                    vm.pageRefocused();
                    found = true;
                }
            }
            if (!found)
            {
                mainViewModel.ViewModel = (ViewModelBase)Activator.CreateInstance(type);
            }
            return found;
        }

        //To be used if a view model needs an input parameter
        protected bool setViewModelByType(Type type, object viewModelInput)
        {
            bool found = false;
            foreach (ViewModelBase vm in ViewModelList)
            {
                if (vm.GetType().Equals(type))
                {
                    mainViewModel.ViewModel = viewModelInput;
                    vm.pageRefocused();
                    found = true;
                }
            }
            if (!found)
            {
                mainViewModel.ViewModel = (ViewModelBase)Activator.CreateInstance(type);
            }
            return found;
        }
    }
}
```
{ mainViewModel.ViewModel = vm; vm.pageRefocused(); found = true; if (type.Equals(typeof(RunExerciseViewModel))) {{RunExerciseViewModel}vm).SetExercise({Exercise}viewModelInput); } } if (!found) { if (type.Equals(typeof(RunExerciseViewModel))) mainViewModel.ViewModel = new RunExerciseViewModel({Exercise}viewModelInput); else mainViewModel.ViewModel = (ViewModelBase)Activator.CreateInstance(type); } return found; }

//Used when a view model is reselected as the content bound ViewModel property protected virtual void pageRefocused() {
 if (ForceDataRead.ForceConnected)
 ForceDataRead.ForceToZero();
}

MainViewModel.cs:

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
namespace URehabAlphaBeta.ViewModels
{
    public class MainViewModel : ViewModelBase
    {
        private ViewModelBase _viewModel;

        public ViewModelBase ViewModel { get { return _viewModel; } set { _viewModel = value; OnPropertyChanged("ViewModel"); } }

        public MainViewModel()
        {
            mainViewModel = this;
            ViewModel = new MainPageViewModel();
        }
    }
}

MainPageViewModel.cs:

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using URehabAlphaBeta.Utilities;
using URehabAlphaBeta.Models;
using System.Diagnostics;
using System.Timers;

namespace URehabAlphaBeta.ViewModels
{
    public class MainPageViewModel : ViewModelBase
    {
        private string _result;
        private string _sendResponse;
        private string _command;
        private bool _isConnected = false;
        private string _forceConnected;

        private string _x;
        private string _y;
        private string _z;
        private string _rx;
        private string _ry;
        private string _rz;

        private Timer zeroingDelayTimer;

        public string X { get { return _x; } set { _x = value; OnPropertyChanged("X"); } }
        public string Y { get { return _y; } set { _y = value; OnPropertyChanged("Y"); } }
        public string Z { get { return _z; } set { _z = value; OnPropertyChanged("Z"); } }
        public string RX { get { return _rx; } set { _rx = value; OnPropertyChanged("RX"); } }
        public string RY { get { return _ry; } set { _ry = value; OnPropertyChanged("RY"); } }
        public string RZ { get { return _rz; } set { _rz = value; OnPropertyChanged("RZ"); } }

        public string Result { get { return _result; } set { _result = value; OnPropertyChanged("Result"); } }
        public string SendResponse { get { return _sendResponse; } set { _sendResponse = value; OnPropertyChanged("SendResponse"); } }
        public string Command { get { return _command; } set { _command = value; OnPropertyChanged("Command"); } }
        public bool IsConnected { get { return _isConnected; } set { _isConnected = value; OnPropertyChanged("IsConnected"); OnPropertyChanged("ConnectionColor"); OnPropertyChanged("ConnectionText"); } }

        public string ConnectionColor { get { if (IsConnected) return "Green"; else return "Red"; } }
        public string ConnectionText { get { if (IsConnected) return "Disconnect"; else return "Connect"; } }
        public string ForceConnected { get { return _forceConnected; } set { _forceConnected = value; OnPropertyChanged("ForceConnected"); } }

        private Timer connectionRetryTimer;

        public MainPageViewModel() : base()
        {
            IsConnected = false;
            Robot.OnSenderResponse += DA_OnSenderResponse;
            Robot.OnDataReceived += DA_OnDataReceived;
            Robot.OnConnected += DA_OnConnected;

            // Checks if force sensor connects successfully and connects to the control and data receive ports on UR5
            if (!Robot.ConnectForceSensor())
            {
                ForceConnected = "Red";
                connectionRetryTimer = new Timer(1000);
                connectionRetryTimer.Elapsed += ConnectionRetryTimer_Elapsed;
                connectionRetryTimer.Start();
            }
            else
            {
                Robot.ConnectReceiver();
                Robot.ConnectSender();
                ForceConnected = "Green";
            }
        }

        // To be run regularly until successful connection
        private void ConnectionRetryTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            if (!Robot.ConnectForceSensor())
            {

            }
        }
    }
}
connectionRetryTimer.Stop();
ForceConnected = "Green";
Robot.ConnectReceiver();
Robot.ConnectSender();
{
//Response received from control port
private void DataAccessor_OnSenderResponse(SenderResponseEventArgs _args)
{
    SendResponse = _args.Response;
}
//Triggers when successfully connected. Turns on digital output port 1 on UR5
private void DataAccessor_OnConnected(EventArgs e)
{
    IsConnected = true;
    Robot.SendCommand("set_digital_out(1,True)" + "\n");
    if(zeroingDelayTimer == null)
    {
        zeroingDelayTimer = new Timer(500);
        zeroingDelayTimer.Elapsed += ZeroingDelayTimer_Elapsed;
    }
    zeroingDelayTimer.Start();
}
//Zeroes the force
private void ZeroingDelayTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    zeroingDelayTimer.Stop();
    zeroingDelayTimer.Close();
    zeroingDelayTimer.Dispose();
    ForceDataRead.ForceToZero();
    zeroingDelayTimer = null;
}
//Receives data from robot and binds to properties. For verification purposes
private void DataAccessor_OnDataReceieved(EventArgs e)
{
    RobotData data = Robot.CurrentData;
    X = "X:" + data.Position.X;
    Y = "Y:" + data.Position.Y;
    Z = "Z:" + data.Position.Z;
    RX = "RX:" + data.Position.RX;
    RY = "RY:" + data.Position.RY;
    RZ = "RZ:" + data.Position.RZ;
}
//Disconnects the sockets
public void DisconnectFromServer()
{
    Robot.StopReceiver();
    Robot.StopSender();
    Robot.DisconnectForceSensor();
    IsConnected = false;
}
//Methods for opening the different views
public void OpenCarGame()
{
    ViewChain.Add(this);
    mainViewModel.ViewModel = new CarGameViewModel();
}
public void OpenCreateExercise()
{
    ViewChain.Add(this);
    setViewModelByType(typeof(MakeExerciseViewModel));
}
public void OpenLoadExercise()
{
    ViewChain.Add(this);
    setViewModelByType(typeof(LoadExerciseViewModel));
}
public void OpenTest()
{
    ViewChain.Add(this);
    setViewModelByType(typeof(TestViewModel));
}
//Turns on digital output port for the force sensor and zeroes the force data
public void Zero()
{
    Robot.SendCommand("set_digital_out(1,True)" + "\n");
    ForceDataRead.ForceToZero();
}

LoadExerciseViewModel.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using URehabAlphaBeta.Utilities;
using System.IO;
using System.Collections.ObjectModel;
using System.Diagnostics;
using URehabAlphaBeta.Models;

namespace URehabAlphaBeta.ViewModels
{
    public class LoadExerciseViewModel : ViewModelBase
    {
        private ObservableCollection<Exercise> _exercises;
        private Exercise _selectedExercise;
        private bool _isSelected;
        private String _feedback;

        public ObservableCollection<Exercise> Exercises { get { return _exercises; } set { _exercises = value; OnPropertyChanged("Exercises"); } }
        public Exercise SelectedExercise { get { return _selectedExercise; } set { _selectedExercise = value; OnPropertyChanged("SelectedExercise"); IsSelected = true; } }
        public bool IsSelected { get { return _isSelected; } set { _isSelected = value; OnPropertyChanged("IsSelected"); } }
        public String Feedback { get { return _feedback; } set { _feedback = value; OnPropertyChanged("Feedback"); } }

        public LoadExerciseViewModel() : base()
        {
            _exercises = new ObservableCollection<Exercise>();
            _exercises.CollectionChanged += _exercises_CollectionChanged;
            _isSelected = false;
            Feedback = "Select an Exercise";
            loadFiles();
        }

        //Sets the RunExerciseViewModel as the content property
        public void OpenRunExercise()
        {
            ViewModelBase.ViewChain.Add(this);
            setViewModelByType(typeof(RunExerciseViewModel), _selectedExercise);
        }

        //Go to previous view
        public void GoBack()
        {
            base.goBack();
        }

        //When the collection changes, it notifies the view
        private void _exercises_CollectionChanged(object sender,
            System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
OnPropertyChanged("Exercises");

// Loads the XML files in the Exercises folder
private void loadFiles()
{
    String[] files = Directory.GetFiles("./Exercises/");
    foreach (String file in files)
    {
        if (file.EndsWith(".xml"))
        {
            try
            {
                addExercise(ExerciseXML.FromXML(file));
            }
            catch (Exception e)
            {
                Debug.Assert(false, e.ToString());
            }
        }
    }
    sortList();
}

// Adds exercise to the _exercises list
private void addExercise(Exercise ex)
{
    if (!exerciseExists(ex))
        _exercises.Add(ex);
}

// Checks if a certain Exercise already exist in the _exercises list
private bool exerciseExists(Exercise ex)
{
    bool exist = false;
    int count = 0;
    while (!exist && count < _exercises.Count)
    {
        if (_exercises[count].Name.Equals(ex.Name))
            exist = true;
        count++;
    }
    return exist;
}

// Sorts the _exercises list alphabetically
private void sortList()
{
    List<Exercise> list = _exercises.ToList();
    list.Sort((x, y) => String.Compare(x.Name, y.Name));
    _exercises = new ObservableCollection<Exercise>(list);
}

// Loads new files when page is refocused
protected override void pageRefocused()
{
    loadFiles();
}

// Sets the feedback property to the selected exercise
public void newSelected(String exerciseName)
{
    foreach (Exercise e in Exercises)
    {
        if (e.Name == exerciseName)
        {
            SelectedExercise = e;
            Feedback = "You Have Selected The Exercise: ";
        }
    }
}
namespace URehabAlphaBeta.ViewModels
{
    public class RunExerciseViewModel : ViewModelBase
    {
        private String _prepareStartText;
        private String _infoText;
        private Exercise _exercise;
        private ExecuteExercise executeExercise;
        private String _currentForceInDirection;
        private String _forceData;
        private String _tangentVector;
        private Double _scoreGage;
        private Color _scoreGageColor;
        private String _numberOfRepetitions;
        private bool calibrated;

        public String PrepareStartText { get { return _prepareStartText; } set { _prepareStartText = value; OnPropertyChanged("PrepareStartText"); } }
        public String InfoText { get { return _infoText; } set { _infoText = value; OnPropertyChanged("InfoText"); } }
        public String CurrentForceInDirection { get { return _currentForceInDirection; } set { _currentForceInDirection = value; OnPropertyChanged("CurrentForceInDirection"); } }
        public String ForceData { get { return _forceData; } set { _forceData = value; OnPropertyChanged("ForceData"); } }
        public String TangentVector { get { return _tangentVector; } set { _tangentVector = value; OnPropertyChanged("TangentVector"); } }
        public String NumberOfRepetitions { get { return _numberOfRepetitions; } set { _numberOfRepetitions = value; OnPropertyChanged("NumberOfRepetitions"); } }

        public Double ScoreGage { get { return _scoreGage; } set { _scoreGage = value; OnPropertyChanged("ScoreGage"); } }
        public Color ScoreGageColor { get { return _scoreGageColor; } set { _scoreGageColor = value; OnPropertyChanged("ScoreGageColor"); } }
        public String NumberOfRepetitions { get { return _numberOfRepetitions; } set { _numberOfRepetitions = value; OnPropertyChanged("NumberOfRepetitions"); } }

        public RunExerciseViewModel(Exercise exercise) : base()
        {
            this._exercise = exercise;
            PrepareStartText = "Prepare Exercise";
            ScoreGage = 0;
            ScoreGageColor = Color.FromRgb(0, 255, 0);
            SetExercise(exercise);
        }

        //Updates the repetition count after each successful repetition
        private void ExecuteExercise_RepetitionsChanged(object sender, EventArgs e)
        {
            OnPropertyChanged("Repetitions");
            NumberOfRepetitions = "Repetitions: " + Repetitions;
        }

        //Sets the exercise to be executed and initializes components
        public void SetExercise(Exercise exercise)
        {
            calibrated = false;
            InfoText = "Press prepare: Keep your distance as the robot gets in to position";
        }
    }
}
executeExercise = new ExecuteExercise(exercise);
executeExercise.StatusChanged += ExecuteExercise_StatusChanged;
executeExercise.RepetitionsChanged += ExecuteExercise_RepetitionsChanged;
NumberOfRepetitions = "Repetitions: " + Repeitions;
ForceDataRead.newData += ForceDataRead_newData;
}

private void ForceDataRead_newData(ForceDataEventArgs e)
{
    Force f = e.ForceData;
    TangentVector = String.Format("Tangent Vector - X: {0} Y:{1} Z:{2}\n", tangent.X, tangent.Y, tangent.Z);
    CurrentForceInDirection = f.ForceInDirection(tangent).ToString();
    if (executeExercise.Status == ExerciseStatus.STARTED)
    {
    ScoreGage = e.ForceData.ForceInDirection(Robot.CurrentData.PositionTangent) * 0.2;
    }
}

private void ExecuteExercise_StatusChanged(object sender, EventArgs e)
{
    if (executeExercise.Status == ExerciseStatus.READY)
    {
    calibrated = false;
    PrepareStartText = "Calibrate";
    InfoText = "Hold on to the handle and press calibrate";
    }
    else if (executeExercise.Status == ExerciseStatus.FINISHED)
    {
    ScoreGage = 0;
    InfoText = "Press Run Again to do the exercise one more time";
    PrepareStartText = "Run Again";
    }
}

private void prepareExercise()
{
    executeExercise.Prepare();
}

private void startExercise()
{
    executeExercise.Start();
}

public void GoBack()
{
    base.goBack();
}
namespace URehabAlphaBeta.ViewModels
{
    class MakeExerciseViewModel : ViewModelBase
    {
        private URMove urMove;
        private List<URMove> points;
        private int _waitTime;
        private String _exerciseName;
        private String _moveType;
        private String _pointInfo;
        private String _feedback;
        private Boolean _setWaitEnabled;
        private Boolean _setAddWaitEnabled;
        private Boolean _setYesNoEnabled;
        private Boolean _setCreateExerciseEnabled;
        private Boolean _linearChecked;
        private Boolean _firstPointInMovement;
        private Boolean _nameSet;
        private Double _x;
        private Double _y;
        private Double _z;

        public String ExerciseName { get { return _exerciseName; } set { _exerciseName = value; _nameSet = true; OnPropertyChanged("ExerciseName"); } }
        public String MoveType { get { return _moveType; } set { _moveType = value; OnPropertyChanged("MoveType"); } }
        public int WaitTime { get { return _waitTime; } set { _waitTime = value; OnPropertyChanged("WaitTime"); } }
        public String PointInfo { get { return _pointInfo; } set { _pointInfo = value; OnPropertyChanged("PointInfo"); } }
        public String Feedback { get { return _feedback; } set { _feedback = value; OnPropertyChanged("Feedback"); } }
        public Boolean SetWaitEnabled { get { return _setWaitEnabled; } set { _setWaitEnabled = value; OnPropertyChanged("SetWaitEnabled"); } }
        public Boolean SetAddWaypointEnabled { get { return _setAddWaitEnabled; } set { _setAddWaitEnabled = value; OnPropertyChanged("SetAddWaypointEnabled"); } }
        public Boolean SetYesNoEnabled { get { return _setYesNoEnabled; } set { _setYesNoEnabled = value; OnPropertyChanged("SetYesNoEnabled"); } }
        public Boolean SetCreateExerciseEnabled { get { return _setCreateExerciseEnabled; } set { _setCreateExerciseEnabled = value; OnPropertyChanged("SetCreateExerciseEnabled"); } }
    }
}
public Boolean LinearChecked { get { return _linearChecked; } set { _linearChecked = value; if (value == true) _moveType = "movep"; else _moveType = "movec"; OnPropertyChanged("LinearChecked"); } }

public Boolean NameSet { get { return _nameSet; } set { _nameSet = value; OnPropertyChanged("NameSet"); } }

public Boolean Curved { get { return _curved; } set { _curved = value; OnPropertyChanged("Curved"); } }

public Double X { get { return _x; } set { _x = value; OnPropertyChanged("X"); } }

public Double Y { get { return _y; } set { _y = value; OnPropertyChanged("Y"); } }

public Double Z { get { return _z; } set { _z = value; OnPropertyChanged("Z"); } }

public MakeExerciseViewModel() : base()
{
    Robot.OnDataReceieved += DataAccessor_OnDataReceieved;

    points = new List<URMove>;
    _waitTimes = new List<int>;
    freeMove = new FreeMove(3);
    _firstPointInMovement = true;
    SetWaitEnabled = false;
    SetYesNoEnabled = false;
    SetCreateExerciseEnabled = false;
    SetAddWaypointEnabled = true;
    LinearChecked = true;
}

private void DataAccessor_OnDataReceieved(EventArgs e)
{
}

//Allows adding more points to the Exercise
public void AddMore()
{
    SetYesNoEnabled = false;
    SetAddWaypointEnabled = true;
    freeMove.Start();
}

//No more points can be added to the Exercise. Activates CreateExercise button
public void DonePoints()
{
    SetYesNoEnabled = false;
    SetCreateExerciseEnabled = true;
}

//adds the current position of the TCP to the list of points for the Exercise
public void AddPoint()
{
    //If the movement is linear, only one point is added
    if (!_curved)
    {
        points.Add(new URMove(Robot.CurrentData.Position, "movep");
        SetWaitEnabled = true;
        SetAddWaypointEnabled = false;
        freeMove.Stop();
    }
    else
    {
        //adds the first point in a curved movement
        if (_firstPointInMovement)
        {
            _firstPointInMovement = false;
        }
        //adds the destination point of a curved movement
        else
        {
            //adds the first point in a curved movement
            if (_firstPointInMovement)
            {
                _firstPointInMovement = false;
            }
            //adds the destination point of a curved movement
            else
            {

        180
```csharp
urMove.AddDestinationPoint(Robot.CurrentData.Position);
points.Add(urMove);
_firstPointInMovement = true;
SetWaitEnabled = true;
SetAddWaypointEnabled = false;
freeMove.Stop();
}
}

// Adds a waiting time to the most recent movement of the exercise
public void AddWaitTime()
{
    _waitTimes.Add(_waitTime);
    SetWaitEnabled = false;
    SetYesNoEnabled = true;
    if (points.Last() != null)
        PointInfo += points.Last().ToString() + "WaitTime: " + _waitTime + "\n";
}

// Creates the exercise and prepares for a test run
public void CreateExercise()
{
    if (_exerciseName != null)
    {
        if (._exerciseName.Equals("")
        {
            try
            {
                ex = new Exercise(points.ToArray(), _waitTimes.ToArray(), _exerciseName);
                exEx = new ExecuteExercise(ex, true, 0.4);
                exEx.StatusChanged += ExEx_StatusChanged;
                exEx.Prepare();
            }
            catch(Exception e)
            {
                String excep = e.StackTrace;
            }
        }
        else
            Feedback = "Missing name";
        else
            Feedback = "Missing name";
    }

    // Test runs the exercise and writes it to xml if test run is successful
    private void ExEx_StatusChanged(object sender, EventArgs e)
    {
        if (exEx.Status == ExerciseStatus.READY)
        {
            Thread.Sleep(1000);
            exEx.Start();
        }
        else if (exEx.Status == ExerciseStatus.FINISHED)
        {
            ExerciseXML.ToXML(ex);
            mainViewModel.ViewModel = new FeedbackViewModel("Exercise Successfully Created", 1000,
            typeof(MainPageViewModel));
            this.close();
        }
    }

    // Stops free move servo and opens previous view
    public void GoBack()
    {
        freeMove.Stop();
        base.goBack();
    }

    // Restarts the free move servo when page is refocused
    protected override void pageRefocused()
    {
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Timers;
namespace URehabAlphaBeta.ViewModels {
    public class FeedbackViewModel : ViewModelBase {
        private String _feedback;
        private Timer timer;
        private Type type;

        public String Feedback { get { return _feedback; } set { _feedback = value; OnPropertyChanged("Feedback"); } }

        public FeedbackViewModel() : base() {
        }

        public FeedbackViewModel(String feedback, int displayTime, Type destination) {
            Feedback = feedback;
            type = destination;
            timer = new Timer(displayTime);
            timer.Elapsed += Timer_Elapsed;
            timer.Start();
        }

        public void SetFeedback(String feedback, int displayTime, Type destination) {
            Feedback = feedback;
            type = destination;
            timer = new Timer(displayTime);
            timer.Elapsed += Timer_Elapsed;
            timer.Start();
        }

        private void Timer_Elapsed(object sender, ElapsedEventArgs e) {
            setViewModelByType(type);
            timer.Stop();
            timer.Close();
        }
    }
}

FeedbackViewModel.cs:
namespace URehabAlphaBeta.ViewModels
{
    public class CarGameViewModel : ViewModelBase
    {
        private Timer updateTimer;
        private int x;
        private int y;
        private const int SPEEDSCALAR = 4000;
        private FreeMove freeMove;
        private Boolean[][] collisionBitMap;
        const int SPEED = 4;
        private int previousX;
        private int previousY;
        private String _speed;
        private String _max;
        private String _speedFarge;
        private String _lapTime;
        private String _bestLap;
        private String _countMargin;
        private String _topSpeedFontWeight;
        private double _maxSpeed;
        private double _curSpeed;
        private double _bestRound;
        private int previousPosX;
        private int previousPosY;
        private SoundPlayer sp;
        private SoundPlayer sp2;
        private SoundPlayer sp3;
        private SoundPlayer sp4;
        private Random generator;
        private bool checkPointCrossed;
        private bool connectedMode;
        private bool freeMoveStopped;
        private bool started;
        private bool playing;
        private int startCount;
    }
}
Timer startTimer;
Timer soundTimer;
Timer freeMoveTimer;
Timer fontSizeTimer;
Stopwatch roundWatch;

private bool xIsIncreasing;
private bool xIsDecreasing;
private bool yIsIncreasing;
private bool yIsDecreasing;

private String _startCount;
private Double _angle;
private String _margin;

public String StartCount { get { return _startCount; } set { _startCount = value; OnPropertyChanged("StartCount"); } } 
public Double Angle { get { return _angle; } set { _angle = value; OnPropertyChanged("Angle"); } } 
public String Margin { get { return _margin; } set { _margin = value; OnPropertyChanged("Margin"); } } 
public String CountMargin { get { return _countMargin; } set { _countMargin = value; OnPropertyChanged("CountMargin"); } } 
public String Speed { get { return _speed; } set { _speed = value; OnPropertyChanged("Speed"); } } 
public String Max { get { return _max; } set { _max = value; OnPropertyChanged("Max"); } } 
public String SpeedFarge { get { return _speedFarge; } set { _speedFarge = value; OnPropertyChanged("SpeedFarge"); } } 
public String LapTime { get { return _lapTime; } set { _lapTime = value; OnPropertyChanged("LapTime"); } } 
public String BestLap { get { return _bestLap; } set { _bestLap = value; OnPropertyChanged("BestLap"); } } 
public String TopSpeedFontWeight { get { return _topSpeedFontWeight; } set { _topSpeedFontWeight = value; OnPropertyChanged("TopSpeedFontWeight"); } } 

public CarGameViewModel()
{
    //Initialize sounds used in the game
    sp = new SoundPlayer("../../Resources/crash.wav");
    sp2 = new SoundPlayer("../../Resources/crash2.wav");
    sp3 = new SoundPlayer("../../Resources/ding.wav");
    sp4 = new SoundPlayer("../../Resources/ding2.wav");

    Tracker trackStart;
    generator = new Random();

    playing = false;
    soundTimer = new Timer(500);
    soundTimer.Elapsed += Timer_Elapsed;

    //initialize round timer
    roundWatch = new Stopwatch();

    started = false;
    _bestRound = 1000;
    checkPointCrossed = false;
    freeMoveStopped = false;

    //Gets bitmap file used for collision detection
    FileStream reader = File.OpenRead("../../Resources/CarTrack5.bmp");

    byte[] buffer = new byte[reader.Length];
    reader.Read(buffer, 0, buffer.Length);

    int MyOffset = BitConverter.ToInt32(buffer, 10);

    byte[] pixelData = new byte[1920 * 1080 * 2 + 56];
    for(int i = MyOffset; i < buffer.Length; i++)
        pixelData[i] = buffer[i];
byte[][] pixels = new byte[1080][];

//Sets the pixel values from the buffer into an array called pixels
for(int i = 0; i < 1080*2 - 1; i++)
{
    if(i%2 == 0)
        pixels[i/2] = new byte[1920];
    for(int j = 0; j < 1920*2; j++)
    {
        if (j % 2 == 0 && i%2 == 0)
            pixels[i / 2][j / 2] = pixelData[(i * 1920) + j];
    }
}

collisionBitMap = new Boolean[1920][];

//Initialize boolean matrix used for collision detection based on the value of the pixel
data. Black pixel == false == collision
for (int i = 0; i < 1920; i++)
{
    collisionBitMap[i] = new Boolean[1080];
    for (int k = 0; k < 1080; k++)
    {
        if (pixels[k][i] == 0)
            collisionBitMap[i][1080 - k - 1] = true;
        else
            collisionBitMap[i][1080 - k - 1] = false;
    }
}

x = 400;
y = 100;

previousX = 0;
previousY = 0;
previousPosX = 0;
previousPosY = 0;

xIsIncreasing = false;
xIsDecreasing = false;
yIsDecreasing = false;
yIsIncreasing = false;

URehabAlphaBeta.Models.Point startPoint = new URehabAlphaBeta.Models.Point(0.5, -0.1, 0.5, 0, 3.14, 0);
URMoveP move = new URMoveP(startPoint);
trackStart = new Tracker();
trackStart.setPointToTrack(startPoint);
trackStart.OnApproachingPoint += TrackStart_OnApproachingPoint;
reader.Close();

Margin = String.Format("{0},{1}" , x, y);
Angle = 90;

CountMargin = "900,300";

startCount = 6;
StartCount = (startCount/2).ToString();
startTimer = new Timer(750);
startTimer.Elapsed += StartTimer_Elapsed;
freeMoveTimer = new Timer(300);
freeMoveTimer.Elapsed += FreeMoveTimer_Elapsed;

fontSizeTimer = new Timer(300);
fontSizeTimer.Elapsed += FontSizeTimer_Elapsed;
if (Robot.SenderConnected) {
    Robot.SendCommand(move);
    connectedMode = true;
} else {
    connectedMode = false;
    startTimer.Start();
}

//Restarts the free move after a collision
private void FreeMoveTimer_Elapsed(object sender, ElapsedEventArgs e) {
    freeMoveStopped = false;
    freeMoveTimer.Stop();
}

//Starts the game start countdown when robot is in initial position
private void TrackStart_OnApproachingPoint(EventArgs e) {
    startTimer.Start();
}

private void FontSizeTimer_Elapsed(object sender, ElapsedEventArgs e) {
    TopSpeedFontWeight = "Normal";
    fontSizeTimer.Stop();
}

//Counts down till start and starts the game
private void StartTimer_Elapsed(object sender, ElapsedEventArgs e) {
    if (startCount % 2 == 1) {
        if (startCount == 1) {
            if (connectedMode)
                freeMove = new FreeMove();
            updateTimer = new Timer(1000 / 120);
            updateTimer.Elapsed += UpdateTimer_Elapsed;
            updateTimer.Start();
            startCount -= 1;
            CountMargin = "800,300";
            StartCount = "GO!";
        } else if (startCount == 0) {
            startTimer.Stop();
            startTimer.Dispose();
            StartCount = "";
        } else {
            startCount -= 1;
            StartCount = (startCount / 2).ToString();
        }
    } else if (startCount == 0) {
        startCount--;
    } else {
        StartCount = "";
        startCount--;
    }
}

//keyboard controls for offline testing
public void xIncrease() {

xIsIncreasing = true;
xIsDecreasing = false;
}
public void xDecrease()
{
    xIsDecreasing = true;
    xIsIncreasing = false;
}
public void yIncrease()
{
    yIsIncreasing = true;
    yIsDecreasing = false;
}
public void yDecrease()
{
    yIsDecreasing = true;
    yIsIncreasing = false;
}
public void xIncStop()
{
    xIsIncreasing = false;
}
public void xDecStop()
{
    xIsDecreasing = false;
}
public void yIncStop()
{
    yIsIncreasing = false;
}
public void yDecStop()
{
    yIsDecreasing = false;
}

//Update timer for the game loop. Used as function for the game logic
private void UpdateTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    //Movement and collision detection
    if (connectedMode)
    {
        int currentPosX = (int)((-Robot.CurrentData.Position.Y) * SPEEDSCALAR);
        if (previousPosX != 0 && previousPosY != 0)
        {
            int changeX = x + (previousPosX - currentPosX) + 40;
            int changeY = y + (currentPosY - previousPosY) + 20;
            if (changeX >= 0 && changeY >= 0)
            {
                if (!collisionBitMap[changeX][changeY])
                {
                    x += previousPosX - currentPosX;
                    y += currentPosY - previousPosY;
                }
                else if (!freeMoveStopped)
                {
                    //Stops robot for 200 milliseconds because of collision
                    freeMove.Stop(200);
                    freeMoveStopped = true;
                    freeMoveTimer.Start();
                    playSound();
                }
            }
        }
    }
    //Movement and collision detection for offline test mode
    else
    {
        if (xIsIncreasing)
        {
            if (x + SPEED < 1920)
            {
if (!collisionBitMap[x + SPEED + 40][y + 20])
    x += SPEED;
else
    playSound();
}
else if (xIsDecreasing)
{
    if (x > SPEED)
    {
        if (!collisionBitMap[x - SPEED + 40][y + 20])
            x -= SPEED;
        else
            playSound();
    }
}
if (yIsIncreasing)
{
    if (y + SPEED < 1080)
    {
        if (!collisionBitMap[x + 40][y + SPEED + 20])
            y += SPEED;
        else
            playSound();
    }
}
else if (yIsDecreasing)
{
    if (y > SPEED)
    {
        if (!collisionBitMap[x + 40][y - SPEED + 20])
            y -= SPEED;
        else
            playSound();
    }
}
// checks if crossing the finish line
crossingTheLine();

int diffX = x - previousX;
int diffY = y - previousY;
if (!started && diffX != 0 || diffY != 0)
{
    started = true;
    roundWatch.Start();
} else
    LapTime = String.Format("Lap Time: {0:00.00}", +(double)roundWatch.ElapsedMilliseconds / 1000);
if (diffX != 0 || diffY != 0)
{
    // Set angle to rotate car based on direction it is traveling in
    Vector diffVec = new Vector(diffX, diffY);
    Angle = Vector.AngleBetween(new Vector(0, 1), diffVec);
    if (diffVec.Length < 99)
        _curSpeed = diffVec.Length;
    Speed = "Speed: " + string.Format("{0:00.00}", _curSpeed);
    if (_curSpeed > _maxSpeed)
    {
        _maxSpeed = _curSpeed;
        sp3.Play();
        Max = "Top Speed: " + string.Format("{0:00.00}", _maxSpeed);
        TopSpeedFontWeight = "UltraBold";
        fontSizeTimer.Start();
    }
}
Margin = String.Format("{0},{1}"), x, y);
//Sets the previous position variables to be used to measure change in the next update.
//Used for speed and angle
if (connectedMode)
{
    previousPosX = (int)(-Robot.CurrentData.Position.Y * SPEEDSCALAR);
    previousX = x;
    previousY = y;
} else
{
    previousX = x;
    previousY = y;
}

//plays the crashing sounds when colliding
private void playSound()
{
    if (!playing)
    {
        playing = true;
        if (generator.NextDouble() > 0.3)
            sp.Play();
        else
            sp2.Play();
        soundTimer.Start();
    }
}

private void Timer_Elapsed(object sender, ElapsedEventArgs e)
{
    soundTimer.Stop();
    playing = false;
}

//checks if player crosses checkpoint or finish line. Checkpoint needs to be crossed before
//finish line
private void crossingTheLine()
{
    if (previousX >= 400 && x <= 400 && y > 0 && y < 400 && started)
    {
        if (checkPointCrossed)
        {
            if (roundWatch.ElapsedMilliseconds/1000 < _bestRound)
            {
                _bestRound = (double)roundWatch.ElapsedMilliseconds / 1000;
                BestLap = String.Format("Best Lap: {0:00.0}", _bestRound);
                sp4.Play();
            }
            checkPointCrossed = false;
            roundWatch.Restart();
        }
        if (x >= 1200 && previousX <= 1200 && y > 600 && y < 1080)
        {
            checkPointCrossed = true;
        }
    }
}

//return to previous screen
public void GoBack()
{
    if (freeMove != null)
        freeMove.Stop();
    updateTimer.Stop();
    this.goBack();
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using URehabAlphaBeta.Utilities;
using URehabAlphaBeta.Models;
using System.Threading;
using System.Diagnostics;
using System.IO;
using System.Globalization;

namespace URehabAlphaBeta.ViewModels
{
    public class TestViewModel : ViewModelBase
    {
        private String _timeInterval;
        private String _force;
        private String _tangent;
        private String _speed;
        private String _calculatedSpeed;
        private String _x;
        private String _y;
        private String _z;
        public String TimeInterval
        {
            get { return _timeInterval; }
            set
            { _timeInterval = value; OnPropertyChanged("TimeInterval"); }
        }
        public String Force
        {
            get { return _force; }
            set { _force = value; OnPropertyChanged("Force"); }
        }
        public String Tangent
        {
            get { return _tangent; }
            set { _tangent = value; OnPropertyChanged("Tangent"); }
        }
        public String Speed
        {
            get { return _speed; }
            set
            { _speed = value; OnPropertyChanged("Speed"); }
        }
        public String CalculatedSpeed
        {
            get { return _calculatedSpeed; }
            set
            { _calculatedSpeed = value; OnPropertyChanged("CalculatedSpeed"); }
        }
        public String X
        {
            get { return _x; }
            set { _x = value; OnPropertyChanged("X"); }
        }
        public String Y
        {
            get { return _y; }
            set { _y = value; OnPropertyChanged("Y"); }
        }
        public String Z
        {
            get { return _z; }
            set { _z = value; OnPropertyChanged("Z"); }
        }
        private List<long> time;
        private List<double> speed;
        private List<double> force;
        private bool written;
        private int count = 0;

        public TestViewModel()
        {
            timer = new System.Timers.Timer(100);
            timer.Elapsed += Timer_Elapsed;
            written = false;
            Robot.OnDataReceieved += Robot_OnDataReceieved1;
            time = new List<long>();
            speed = new List<double>();
            force = new List<double>();
        }
    }
}
private void Robot_OnDataReceieved1(EventArgs e)
{
    if (count == 10)
    {
        Force f = new Force(ForceDataRead.ForceData);
        double force = f.ForceInDirection(Robot.CurrentData.PositionTangent);
        Force = force.ToString();
        X = "FX: " + f.ForceVector.X.ToString();
        Y = "FY: " + f.ForceVector.Y.ToString();
        Z = "FZ: " + f.ForceVector.Z.ToString();
        double speed = force * 0.0005;
        CalculatedSpeed = speed.ToString();
        count = 0;
    }
    count++;
}

private void Timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
{
    double speed = force * 0.0005;
    if (speed > 0.3)
    speed = 0.3;
    else if (speed <= 0)
    speed = 0.001;
    Robot.SendCommand(new URMoveP(new Point(0.3, 0.5, 0.3, 0, 3.14, 0), speed));
}

public void RunWithTimeInterval()
{
    int number;
    if (int.TryParse(_timeInterval, out number))
    {
        Robot.SendCommand(new URMoveP(new Point(0.3, 0.4, 0.3, 0, 3.14, 0)));
        Thread.Sleep(number);
        Robot.SendCommand("stop1(1000)" + "\n");
    }
}

public void DoubleCommands()
{
}

public void GoBack()
{
    this.goBack();
}

public void PrepareMeasurement()
{
    Robot.SendCommand(new URMoveP(new Point(0.3, 0.0, 0.3, 0, 3.14, 0), 0.2));
    double force = new Force(ForceDataRead.ForceData).ForceInDirection(Robot.CurrentData.PositionTangent);
}

public void DataForceMeasurement()
{
    Robot.SendCommand(new URMoveP(new Point(0.3, 0.5, 0.3, 0, 3.14, 0), 0.001));
    watch.Start();
    timer.Start();
    Robot.OnDataReceieved += Robot_OnDataReceieved;
}

private void Robot_OnDataReceieved1(EventArgs e)
{
    if (watch.ElapsedMilliseconds < 10000)
    {

time.Add(watch.ElapsedMilliseconds);
speed.Add(Robot.CurrentData.TcpSpeed);

    double forceInDir = new Force(ForceDataRead.ForceData).ForceInDirection(Robot.CurrentData.PositionTangent);
    force.Add(forceInDir);
}

else if (!written)
    writeToFile();
}

private void writeToFile()
{
    StreamWriter file = new StreamWriter("TestFileExcelGorilla.csv");
    int counter = 0;
    while (counter < time.Count)
    {
            
        counter++;
    }
    written = true;
}

___________________________________________________________________________

Views

App.xaml:

<Application x:Class="URehabAlphaBeta.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:ViewModels="clr-namespace:URehabAlphaBeta.ViewModels"
    xmlns:Views="clr-namespace:URehabAlphaBeta.Views"
    StartupUri="MainWindow.xaml">
    <Application.Resources>
        <DataTemplate DataType="{x:Type ViewModels:MainPageViewModel}"
            ViewName="MainView"/>
        <DataTemplate DataType="{x:Type ViewModels:MakeExerciseViewModel}"
            ViewName="MakeExerciseView"/>
        <DataTemplate DataType="{x:Type ViewModels:LoadExerciseViewModel}"
            ViewName="LoadExerciseView"/>
        <DataTemplate DataType="{x:Type ViewModels:RunExerciseViewModel}"
            ViewName="RunExerciseView"/>
        <DataTemplate DataType="{x:Type ViewModels:FeedbackViewModel}"
            ViewName="FeedbackView"/>
        <DataTemplate DataType="{x:Type ViewModels:TestViewModel}"
            ViewName="TestView"/>
        <DataTemplate DataType="{x:Type ViewModels:CarGameViewModel}"
            ViewName="CarGameView"/>
    </Application.Resources>
    <Style x:Key="RoundedButtonStyle" TargetType="{x:Type Button}">
        <Setter Property="Template">
            <Setter.Value>
<ControlTemplate TargetType="Button">
  <Border CornerRadius="15" BorderThickness="1" Padding="40,10" Width="auto">
    <Border.Background>
      <ImageBrush ImageSource="/URRehabAlphaBeta;component/Resources/ButtonBackground.png"/>
    </Border.Background>
    <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
  </Border>
  <ControlTemplate.Triggers>
    <Trigger Property="IsEnabled" Value="False">
      <Setter Property="Opacity" Value="0.2"/>
    </Trigger>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter Property="FontSize" Value="52"/>
    </Trigger>
    <Trigger Property="IsPressed" Value="True">
      <Setter Property="Opacity" Value="0.8"/>
    </Trigger>
  </ControlTemplate.Triggers>
</ControlTemplate>
</Window>

MainView.xaml:

  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:local="clr-namespace:URRehabAlphaBeta.Views"
  mc:Ignorable="d"
  Content="{Binding Path=ViewModel}"
  Title="MainView" Height="1080" Width="1920" WindowStyle="None" WindowState="Maximized">
  <Grid>
    <Grid Width="900" HorizontalAlignment="Left">
      <StackPanel Orientation="Vertical">
        <!-- Your content here -->
      </StackPanel>
    </Grid>
  </Grid>
</Window>
LoadExerciseView.xaml:

```xml
<Page x:Class="URehabAlphaBeta.Views.LoadExerciseView"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
     xmlns:local="clr-namespace:URehabAlphaBeta.Views"
     mc:Ignorable="d">
  <Grid d:DesignHeight="1880" d:DesignWidth="1920">
    <Button Style="{DynamicResource RoundedButtonStyle}" x:Name="button_TrackPoint" Content="Let's Start An Exercise!" HorizontalAlignment="Stretch" Margin="40,80,40,0" Height="100" VerticalAlignment="Top" Click="button_TrackPoint_Click"/>
    <Button Style="{DynamicResource RoundedButtonStyle}" x:Name="button_LoadExercise" Content="I Just Want To Play!" HorizontalAlignment="Stretch" Margin="40,80,40,0" Height="100" VerticalAlignment="Top" Click="button_LoadExercise_Click"/>
    <Button Style="{DynamicResource RoundedButtonStyle}" x:Name="button_CreateExercise" IsEnabled="{(Binding ElementName=advancedCheckbox, Path=IsChecked)}" Content="Create Exercise" HorizontalAlignment="Stretch" Margin="40,40,40,0" Height="100" VerticalAlignment="Bottom" Click="button_CreateExercise_Click"/>
    <Button Style="{DynamicResource RoundedButtonStyle}" x:Name="button_ShowStatistics" IsEnabled="{(Binding ElementName=advancedCheckbox, Path=IsChecked)}" Content="ShowStatistics" HorizontalAlignment="Stretch" Margin="40,40,40,0" Height="100" VerticalAlignment="Bottom" Click="button_ShowStatistics_Click"/>
    <CheckBox x:Name="advancedCheckbox" Margin="50,0,0,20" Content="Advance Mode" IsChecked="False" VerticalAlignment="Bottom" HorizontalAlignment="Left"/>
  </StackPanel>
</Grid>
</Page>
```
RunExerciseView.xaml:
MakeExerciseView.xaml:

  <Page.Resources>
    <Storyboard x:Key="BackButtonHover" Storyboard.TargetProperty="(UIElement.Opacity)"
      Storyboard.TargetName="image_home">
      <EasingDoubleKeyFrame KeyTime="0:0:0" Value="1"/>
    </Storyboard>
    <Storyboard x:Key="BackButtonLeave" Storyboard.TargetProperty="(UIElement.Opacity)"
      Storyboard.TargetName="image_home">
      <EasingDoubleKeyFrame KeyTime="0" Value="0.5"/>
    </Storyboard>
  </Page.Resources>
  <Grid>
    <Border BorderBrush="#FFD80E0E" BorderThickness="3" CornerRadius="20"
      Margin="40,40,40,0">
      <SolidColorBrush Color="#FFFAEFF" Opacity="0"/>
    </Border>
    <StackPanel Orientation="Vertical" HorizontalAlignment="Right" Height="1080"
      VerticalAlignment="Top" Width="960">
      <Grid Height="300">
        <Border BorderBrush="#FFD80E0E" BorderThickness="3" CornerRadius="20"
          Margin="40,40,40,0">
          <SolidColorBrush Color="White" Opacity="0.6"/>
        </Border>
      </Grid>
      <Grid>
        <ScrollBar Orientation="Horizontal" VerticalAlignment="Top"
          Grid.Column="1" Grid.Row="1" Grid.RowSpan="2" HorizontalAlignment="Stretch"
          Margin="100,0,0,0">
          <Grid.ColumnSpan="3"/>
          <Grid.RowSpan="3"/>
          <SolidColorBrush Color="/URehabAlphaBeta.Views/clr-namespace:URehabAlphaBeta.Views;rid=Brushes;target="Normal" Opacity="0.5"/>
          <StackPanel Orientation="Vertical" Grid.Column="0" Grid.Row="0">
            <TextBlock x:Name="progressLabel" FontSize="20" HorizontalAlignment="Center" Margin="40,0,0,0" VerticalAlignment="Top"></TextBlock>
          </StackPanel>
          <ScrollBar.ValueProperty="changedValue"/>
          <ScrollBarThumb Content="..."/>
        </ScrollBar>
      </Grid>
    </StackPanel>
  </Grid>
</Page>
available exercises later in the execute exercise window." Foreground="White"
RenderTransformOrigin="0.5,0.5"/>
</Border>
<Image x:Name="image_back" Source="/URehabAlphaBeta;component/Resources/BackArrow.png"
HorizontalAlignment="Left" Height="100" Margin="28,11,0,0" VerticalAlignment="Top" Width="100"
MouseLeftButtonUp="image_MouseLeftButtonUp" />
<Image x:Name="image_home" Source="/URehabAlphaBeta;component/Resources/HomeButton.png"
HorizontalAlignment="Left" Height="100" Margin="155,10,0,0" VerticalAlignment="Top" Width="100"/>
</Grid>

FeedbackView.xaml:

(Page x:Class="URehabAlphaBeta.Views.FeedbackView"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:local="clr-namespace:URehabAlphaBeta.Views"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="300"
Title="FeedbackView">
  <Grid>
    <Label x:Name="label" Content="{Binding Path=Feedback}" HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="30"/>
  </Grid>
</Page>

CarGameView.xaml:

(Page x:Class="URehabAlphaBeta.Views.CarGameView"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:local="clr-namespace:URehabAlphaBeta.Views"
mc:Ignorable="d"
d:DesignHeight="1080" d:DesignWidth="1920"
Loaded="Page_Loaded"
Title="CarGameView">
  <Page.Background>
    <ImageBrush ImageSource="/Resources/CarTrack-Evo.jpg"/>
  </Page.Background>
  <Canvas>
    <Image x:Name="image_back" Source="/URehabAlphaBeta;component/Resources/BackArrow.png"
HorizontalAlignment="Left" Height="100" Margin="28,11,0,0" VerticalAlignment="Top" Width="100"
MouseLeftButtonUp="image_back_MouseLeftButtonUp" />
    <StackPanel Orientation="Horizontal">
      <Label Name="SpeedLabel" Content="{Binding Path=Speed}" FontSize="35" Margin="200,0,0,0"/>
      <Label Name="MaxSpeedLabel" FontWeight="{Binding Path=TopSpeedFontWeight}" Content="{Binding Path=Max}" FontSize="35" Margin="50,0,0,0" Foreground="White"/>
      <Label Name="LapTimeLabel" Content="{Binding Path=LapTime}" FontSize="35" Margin="200,0,0,0"/>
      <Label Name="BestLapLabel" Content="{Binding Path=Bestlap}" FontSize="35" Margin="20,0,0,0" Foreground="LightCoral"/>
      <Label Name="StartCount" Content="{Binding Path=StartCount}" FontSize="200" FontWeight="Bold" Foreground="White" HorizontalAlignment="Center" VerticalAlignment="Center" Margin="{Binding Path=CountMargin}"/>
    </StackPanel>
  </Canvas>
</Page>

198
LoadExerciseView.xaml.cs:

```csharp
using URehabAlphaBeta.Utilities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
```

TestView.xaml:

```xml
<Rectangle HorizontalAlignment="Left" RenderTransformOrigin="0.5,0.5" Height="50"
Margin="[Binding Path=Margin]" Stroke="Black" VerticalAlignment="Top" Width="30" StrokeThickness="0">
    <Rectangle.RenderTransform>
        <RotateTransform Angle="{Binding Path=Angle}"></RotateTransform>
    </Rectangle.RenderTransform>
    <Rectangle.Fill>
        <ImageBrush ImageSource="/Resources/RehabCarURLightBluel.png"/>
    </Rectangle.Fill>
</Rectangle>
</Canvas>
</Page>

TestView.xaml:

```xml
<Grid>
    <TextBox x:Name="textBox" HorizontalAlignment="Left" Height="23" Margin="48,66,0,0"
TextWrapping="Wrap" Text="{Binding Path=TimeInterval}"
VerticalAlignment="Top" Width="120"/>
    <Button x:Name="button" Content="Execute" HorizontalAlignment="Left" Margin="173,69,0,0"
VerticalAlignment="Top" Width="75" Click="button_Click"/>
    <Button x:Name="button_Prepare" Content="Prepare" HorizontalAlignment="Left"
Margin="48,130,0,0" VerticalAlignment="Top" Width="75" Click="button_Prepare_Click"/>
    <Button x:Name="button_Start" Content="Start" HorizontalAlignment="Left" Margin="173,130,0,0"
VerticalAlignment="Top" Width="75" Click="button_Start_Click"/>
    <Button x:Name="button_GoBack" Content="Back" HorizontalAlignment="Left" Margin="48,270,0,0"
VerticalAlignment="Top" Width="75" Click="button_GoBack_Click"/>
    <Label x:Name="label" Content="{Binding Path=Force}" HorizontalAlignments="Left"
Margin="48,172,0,0" VerticalAlignment="Top" Width="200" FontSize="20"/>
    <Label x:Name="label1" Content="{Binding Path=Tangent}" HorizontalAlignment="Left"
Margin="48,220,0,0" VerticalAlignment="Top" FontSize="20"/>
    <Label x:Name="label2" Content="{Binding Path=Speed}" HorizontalAlignment="Left"
Margin="48,24,0,0" VerticalAlignment="Top"/>
    <Label x:Name="label3" Content="{Binding Path=CalculatedSpeed}" HorizontalAlignment="Left"
Margin="48,24,0,0" VerticalAlignment="Top"/>
    <Label x:Name="label4" Content="{Binding Path=X}" HorizontalAlignment="Left"
Margin="272,71,0,0" VerticalAlignment="Top"/>
    <Label x:Name="label5" Content="{Binding Path=Y}" HorizontalAlignment="Left"
Margin="272,102,0,0" VerticalAlignment="Top"/>
    <Label x:Name="label6" Content="{Binding Path=Z}" HorizontalAlignment="Left"
Margin="272,133,0,0" VerticalAlignment="Top"/>
</Grid>
</Page>

LoadExerciseView.xaml.cs:
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;
using URehabAlphaBeta.ViewModels;

namespace URehabAlphaBeta.Views
{
    /// <summary>
    /// Interaction logic for LoadExerciseView.xaml
    /// </summary>
    public partial class LoadExerciseView : Page
    {
        public LoadExerciseView()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            ((LoadExerciseViewModel)this.DataContext).OpenRunExercise();
        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            ((LoadExerciseViewModel)this.DataContext).GoBack();
        }

        private void Border_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
        {
            ((LoadExerciseViewModel)this.DataContext).newSelected(((Border)sender).Tag.ToString());
        }

        private void image_back_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {
            ((LoadExerciseViewModel)this.DataContext).GoBack();
        }
    }
}

RunExerciseView.xaml.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;
using URehabAlphaBeta.ViewModels;

namespace URehabAlphaBeta.Views
{
    /// <summary>
    /// Interaction logic for RunExerciseView.xaml
    /// </summary>
    public partial class RunExerciseView : Page
    {
        public RunExerciseView()
        {
            InitializeComponent();
        }
    }
}
private void button_PrepareStart_Click(object sender, RoutedEventArgs e)
{
    ((RunExerciseViewModel)this.DataContext).StartPrepare();
}

private void image_back_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
    ((RunExerciseViewModel)this.DataContext).GoBack();
}

namespace URehabAlphaBeta.Views
{
    public partial class MakeExerciseView : Page
    {
        public MakeExerciseView()
        {
            InitializeComponent();
        }

        private void waypointButton_Click(object sender, RoutedEventArgs e)
        {
            ((MakeExerciseViewModel)this.DataContext).AddPoint();
        }

        private void timeToWaitButton_Click_1(object sender, RoutedEventArgs e)
        {
            ((MakeExerciseViewModel)this.DataContext).AddWaitTime();
        }

        private void button_YES_Click(object sender, RoutedEventArgs e)
        {
            ((MakeExerciseViewModel)this.DataContext).AddMore();
        }

        private void button_NO_Click(object sender, RoutedEventArgs e)
        {
            ((MakeExerciseViewModel)this.DataContext).DonePoints();
        }

        private void button_CreateExercise_Click(object sender, RoutedEventArgs e)
        {
            ((MakeExerciseViewModel)this.DataContext).CreateExercise();
        }
    }
}
private void button_goBack_Click(object sender, RoutedEventArgs e)
{
    ((MakeExerciseViewModel)this.DataContext).GoBack();
}

private void image_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
    ((MakeExerciseViewModel)this.DataContext).GoBack();
}

namespace URehabAlphaBeta.Views
{
    /// <summary>
    /// Interaction logic for CarGameView.xaml
    /// </summary>
    public partial class CarGameView : Page
    {
        public CarGameView()
        {
            InitializeComponent();
        }

        private void Page_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.W)
            {
                ((CarGameViewModel)this.DataContext).yDecrease();
            }
            else if (e.Key == Key.S)
            {
                ((CarGameViewModel)this.DataContext).yIncrease();
            }
            else if (e.Key == Key.A)
            {
                ((CarGameViewModel)this.DataContext).xDecrease();
            }
            else if (e.Key == Key.D)
            {
                ((CarGameViewModel)this.DataContext).xIncrease();
            }
        }

        private void Page_KeyUp(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.W)
            {
                ((CarGameViewModel)this.DataContext).yDecStop();
            }
        }
    }
}
private void Page_Loaded(object sender, RoutedEventArgs e)
{
    var window = Window.GetWindow(this);
    window.KeyDown += Page_KeyDown;
    window.KeyUp += Page_KeyUp;
}

private void image_back_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
    ((CarGameViewModel)thisDataContext).GoBack();
}

TestView.xaml.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;
using URehabAlphaBeta.ViewModels;
namespace URehabAlphaBeta.Views
{
    /// <summary>
    /// Interaction logic for TestView.xaml
    /// </summary>
    public partial class TestView : Page
    {
        public TestView()
        {
            InitializeComponent();
        }

        private void button_Click(object sender, RoutedEventArgs e)
        {
            ((TestViewModel)this.DataContext).RunWithTimeInterval();
        }

        private void button_Prepare_Click(object sender, RoutedEventArgs e)
        {
            ((TestViewModel)this.DataContext).PrepareMeasurement();
        }
    }
}
private void button_Start_Click(object sender, RoutedEventArgs e)
{
    ((TestViewModel)this.DataContext).DataForceMeasurement();
}

private void button_GoBack_Click(object sender, RoutedEventArgs e)
{
    ((TestViewModel)this.DataContext).GoBack();
}