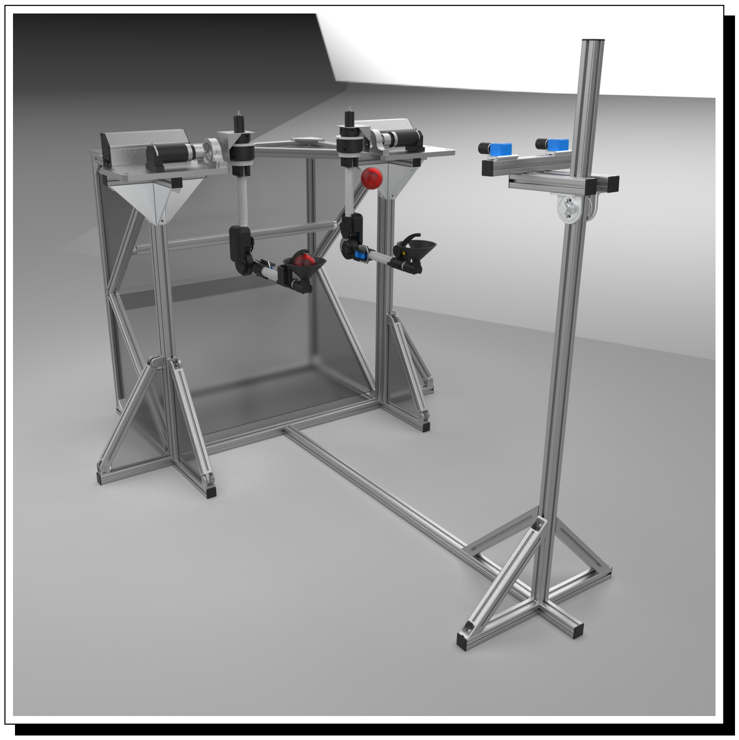
KTH Mechatronics Advanced Course
MF2059, HT 2020
FINAL REPORT

Juggling Robot Project



JuRP20

Faiza Ali, Fredrika Ardestam, Andreas Baldhagen, Simon Calminder,
Tobias Ekman, Filip Elander, Mattias Klang, Behnam Yosef Nezhad Arya

December 13, 2020

Abstract

A mechatronic project is at its best when it combines many fields of study, and the project described in this report does exactly that. Constructing a juggling robot is a feat that requires knowledge in areas such as mechanical construction, electronics, software, communication, and many more.

The JuRP-project is a recurring juggling robot project that this year reaches its third iteration. The goals of this year's project focuses on constructing a juggling robot with two arms that could perform collaborative juggling. Additionally, the system should also be mobile, meaning that no part of the system can be a permanent installation. The solution presented in this report consists of a new set of arms. Last year's arm was redesigned to better meet this year's requirements and to make the system more robust. Also, a second arm was built from scratch, in parallel with redesigning the old arm. Furthermore, a new vision system was designed. This year's new vision system was designed with a mobile rack, differing from the previous permanently installed camera setup. The project was regarded as a success, as all of the stakeholder's requirements and goals were met. The robot achieved a human-like juggling with up to two balls, and will serve as a robust base for future development was established.

Acknowledgement

The JuRP group would like to acknowledge and give extra thanks to the following individuals: First, we would like to thank our coach Daniel Frede and stakeholder Martin Törngren, for great coaching, support, and guidance throughout this project. Secondly, we would like to thank Staffan Qvarnström, Tomas Östberg, and Jan Stamer for all their knowledge and support, and advice during the manufacturing required to complete this project. Another sign of gratitude goes to our sponsors: Kevin Nunkoosingh and Fredrik Ståhl at Compotech for providing us with state of the art servo drivers (Ingenia Pluto). Also, Klas Bengtsson at Recab for lending us the cameras, the level of this project wouldn't have been possible without their help. A final gratitude is directed to the 2019 JuRP group member, Jon Arrizabalaga. Jon was of great value in making it an easy transition from last year's project.

JuRP group 2020
Stockholm, December 2020

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

CAD	Computer Aided Design
CAN	Controller Area Network
CPP	CPP files are source code files within the C++ programming language
DOF	Degrees Of Freedom
fps	frames per second
HSV	Hue, Saturation, Value
$JuRP$	Juggling Robot Project
PID	Proportional Integral Derivative
PLA	Polylactic Acid, PLA plastics are often used in 3D printing
PWM	Pulse Width Modulation
RGB	Red, Green, Blue
ROS	Robot Operating System
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

Other Symbols

$O()$	Execution time required, this notation is used in Computer Science to describe the performance or complexity of an algorithm

Physical Quantities

U	Voltage expressed in volt $[V]$

Chapter 1

Introduction

1.1 Background

Juggling is considered a reasonably simple task for a human, and it usually doesn't take more than a couple of hours for a person to juggle tree balls or more. Humans accomplish this by using a sophisticated machine, also known as the brain, to perform the needed motion control and the eyes to track the ball.

When observing a human or animal moving and coordinating within its environment it often looks simple and straightforward. However, as it is widely known, such apparently simple coordination can be a very complex task in robotics. Achieving and performing juggling is therefore considered a perfect task from a mechatronics- and robotics research perspective, demanding knowledge and performance in a wide range of areas, such as robotic design, sensing, perception, controlling and executing movements and commands, software architecture etc.

The purpose of this capstone-project was to explore and develop a juggling robot, and was done in collaboration with KTH. The JuRP-project has been ongoing for two years and the project was initiated in 2017, making this year's project the third iteration. Each year the project problem statement has been expanded and new goals have been added. Previous work from the most recent project iteration, JuRP-2019, is briefly explained below (section[1.1.1]). For more detailed information of the past project, the reader can find the report in the references [3].

1.1.1 JuRP 2019

The JuRP-2019-project ended with the development and experimentation with one single robotic arm, see Figure 1.1. Many of the precedent project's requirements and goals will be kept for this year as well, such as performing human-like juggling patterns, use external perception, and achieving real-time feedback.

For their vision system, they used permanently installed IR-cameras, making the installment permanent. Figure 1.2 depicts last year's end-effector concepts. The final prototype of JuRP 2019 used the solution with a cone for end-effector, making it unable to grip or hold on to a ball, but very efficient in terms of design complexity and center of mass placement, making it more robust.

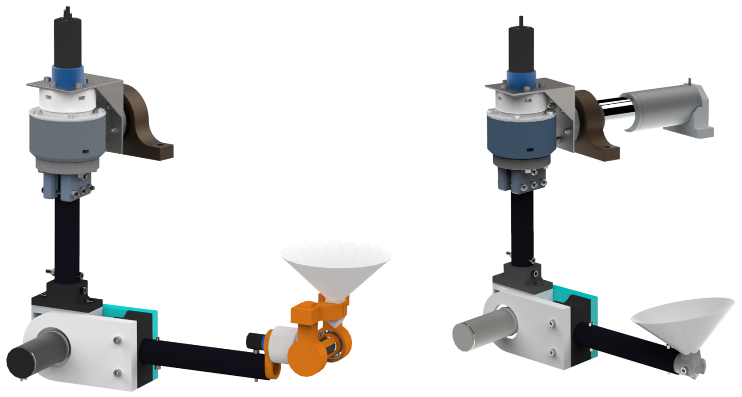


Figure 1.1: CAD of JuRP 2019's robot arm, with two different solutions for end-effector.

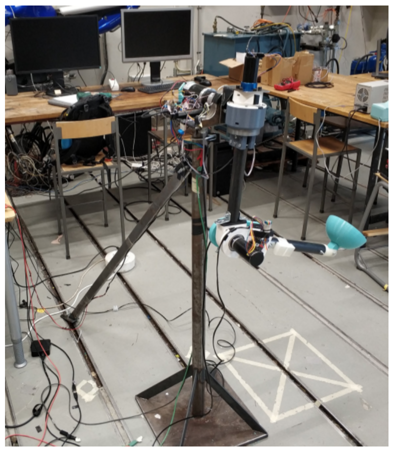


Figure 1.2: End product of the 2019 project.

1.2 Goals for JuRP 2020

Performance Goals

The purpose of the project is to develop a juggling robot consisting of two arms capable of juggling by reliably catching and throwing balls. More specifically:

- Design and construct a functional prototype for a juggling robot.
- Performing human-like autonomous juggling.
- Juggling with 2-3 balls in the air.
- Full system mobility, meaning no permanent installations should be needed.
- Being able to grip balls.

Project Goals

Tasks needed for achieving the above performance goals are stated as the following:

- Achieve collaborative juggling with two robot arms.
- Implement the second arm by utilizing the existing JuRP robot arm and develop the arm to an extent fulfilling necessary performance requirements and then replicate the arm.
- Design and implement a mobile camera system in order to make the system mobile.
- Design and construct an "end-effector" or "gripper" to facilitate throwing and catching.
- Emphasize a well-planned system-architecture with well-defined interfaces, system robustness and facilitated use/maintenance.

A successful project should lead to a stable and robust foundation for coming students or researchers. A variety of research and optimization areas would be available by succeeding in the creation of two functioning real-time juggling robotics arms. Autonomous juggling touches many fields regarding real-time response, feedback systems, and electromechanical design. Countless opportunities for studies exist in all of those areas, especially for master's level students. The project also serves as an excellent learning opportunity for the current students, independent of its success. A failure in reaching performance goals could still lead to success in obtaining academic learning goals.

1.3 Requirements

The project's stakeholders had a few requirements in addition to the goals. The requirements were meant as a guideline for the group's goals, for the project not to diverge into a machine that blindly throws balls without any feedback.

1.3.1 Stakeholder Requirements

- Have human-like juggling pattern and focus on toss juggling on balls
- At a minimum be able to handle one ball, but expected to perform a juggling pattern with 2-3 balls
- Ideally, design and integrate a gripper to allow in-hand manipulation to facilitate 3(or more) ball juggling

1.3.2 Technical Requirements

- Use a mobile vision system for external perceptional (real-time feedback)
- Be robust and prepared for long term evolution
- Emphasizing a well-planned system architecture (including well defined interfaces), system robustness and facilitate use/maintenance

1.4 Delimitations

Due to the possible wide range and extent of a juggling robot project such as this one, the project team has set up some limitations to narrow the scope and make the goals more achievable and the work clearer. The project statement goals have also been divided into phases, making it easier to focus and finish specific milestones if some of the goals would have to be taken out of the scope due to time limitations.
Mainly the scope of the project is:

- Juggling with up to 3 balls with one basic juggling pattern - a bonus if more generalized
- A limited extent of capability for collaborating with humans
- Autonomous juggling - even though start and stop may require manual human intervention depending on the designed gripper's performance and capability

Also, with the uncertainty following the ongoing pandemic at the time of writing the goals, it was not assured that the project team would gain full access to workshops and lab halls during the project. Therefore, the milestones were set in a way to serve as an end-goal if a lockdown would occur.

1. *Milestone A* - Single arm juggling with 1-2 balls, using human throw-catch.
2. *Milestone B* - Extending to juggling with 2-3 balls, using gripper throw-catch.
3. *Milestone C* - Dual-arm juggling with up to 3 balls, possibility of using both human and gripper while also have full system mobility.

1.5 Report Disposition

Chapter 1 explains the background and the purpose of the project, as well as describing the project in terms of goals and requirements.

Chapter 2 consists of the "state of the art", which explains the industry's current technological level and some of the solutions that will be covered in the report.

Chapter 3 presents which methods and tools were used when solving the project problems.

Chapter 4 presents the different design alternatives considered during the design phase, and also discuss the different alternatives.

Chapter 5 covers the final design and implementation of the project. It will present and explain the chosen design and motivate for its use.

Chapter 6 briefly explains the guidelines followed in terms of ethics and environmental impact.

Chapter 7 presents the methods used to verify and validate the requirements stated for the project.

Chapter 8 presents the findings of Chapter 7; results regarding the requirements and also general findings that are relevant to the project result.

Chapter 9 will summarise the findings and discuss the potential success of failure of different parts of the project, as well as the project as a whole.

Chapter 10 suggests any future improvements that can be done or potential developments of the system/parts of the system that can be made in order to get better or an expanded number of functions.

Appendix contains any additional information that was not necessary to have in the main text or information in the form of code or images that would have interrupted the flow of the report.

Chapter 2

Literature Review and State of the Art

The focus in this literature review has been on researching areas which are relevant to the construction of a robot performing juggling. Thus this chapter has been divided into separate sections, which was deemed especially interesting for the project. First, this chapter will briefly dive into the art of juggling. Following this, the report will go through all the different subsystems in which it was deemed necessary to conduct research into possible solutions. Those areas are: Vision System, Software, Controllers, Mechanics and End Effector

2.1 Juggling

There are many different ways to juggle. To get a perspective, the *Library of Juggling* states at least 174 juggling techniques [34] where 136 is three-ball patterns, 29 four-ball patterns, 8 five-ball patterns and 1 six-ball pattern. On top of the number of balls and patterns, all of these can also be varied and combined to an endless number of patterns to be performed by a juggling robot.

There are three main methods of juggling: toss, bounce and contact juggling. In toss juggling a number of objects, called *props* are thrown into the air and caught in some sequence, with the requirement that at least one prop at any given time is in the air. Bounce juggling is like toss juggling except the balls are thrown downwards and bounced back up. The least common method is contact juggling where the juggler rolls the props over their body without throwing them, always having contact with the props. In this project the most common type of juggling will be used: toss juggling with spherical props. The starting pattern in toss juggling for most beginners is the 3 ball cascade also called *siteswap 3*. A good way to learn this is by starting with one ball, then two and finally three balls. Figure 2.1 shows how the props are thrown from one hand to the other always swapping hands and creating a horizontal figure-eight trace.

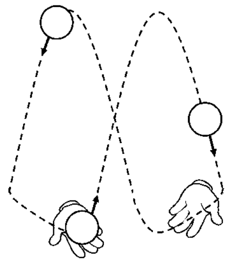
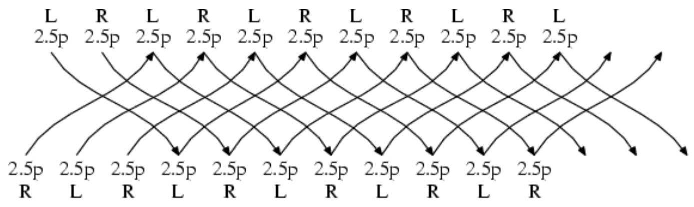


Figure 2.1: The 3 ball cascade pattern.

The number of props can be lowered to two if only one hand is used, however this puts more emphasis on timing due to the balls spending less time in the hand. Just as the number of props can be changed so can the number of jugglers. In cooperative juggling, the patterns are described by how often the props are passed between the jugglers; if every throw is a pass it is called 1-count, every second throw being a pass would be a 2-count and so on. One common pattern in two person juggling is the six ball 1-count, basically two separate three ball cascade patterns between the jugglers. Another way would simply be a replica of the 3 ball cascade but each juggler only using one hand. The five ball 1-count is the easiest dual-hand cooperative juggling pattern where one juggler throws the balls straight forward and the other juggler throws diagonally. Juggling patterns can be described mathematically by ladder diagrams, Figure 2.2 shows how the top juggler passes diagonally and the bottom juggler passes straight, e.g. their right hand to the other juggler's left hand.



L R L R L R L R L R L
2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p
2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p 2.5p
R L R L R L R L R L R

Figure 2.2: The 5 ball 1-count cooperative juggling pattern.

2.1.1 Juggling Robots

The first juggling robots, relying on an open loop bouncing construction, were built by Claude Shannon in the 1970s, shown in Figure 2.3 [12]. Shannon's work inspired others to investigate the challenges provided by the task of juggling to showcase the precision and repetitiveness of robotics and automation and to explore areas of different fields. This include areas of machine learning and human behavior by Atkenson et al. [2] [4].

Figure 2.3: Bounce juggling robot made by Claude Shannon in the 70s.

Many have expanded the task by adding a closed loop design for better robustness towards variance in the mechanical construction and to improve the performance [29]. Since the 1990s, several toss juggling robots have been successfully designed based on feedback from a computer vision system such as Rizzi et al. [26] and Yu et al. [36] and potentially giving the robot the ability to participate in cooperative juggling. However, there are few examples of human-machine juggling.

One particularly successful juggling robot was developed at the Czech Technical University in Prague capable of juggling five balls using a vertical sliding mechanism [10]. Below is a presentation of the design of a humanoid robot and a humanoid arm, that are more in line with the scope of this project than the Czech robot.

In 2017 Tomoki Oka, Naohiro Komura and Akio Namiki at Chiba University in Japan solved the task of juggling by developing a human-like multi fingered hand-arm robot, with a high speed stereo vision system, capable of juggling two balls. Figure 2.4 shows the seven DOF arm and the three fingered high speed end effector with two DOF for the index finger and 3 DOF for the other two fingers. For tracking the balls the high speed stereo vision system platform IDP express [14] was used running at 900 Hz and the two cameras were placed in two different planes to capture the movement in 3D space. The position of the balls were obtained by color extraction [23].

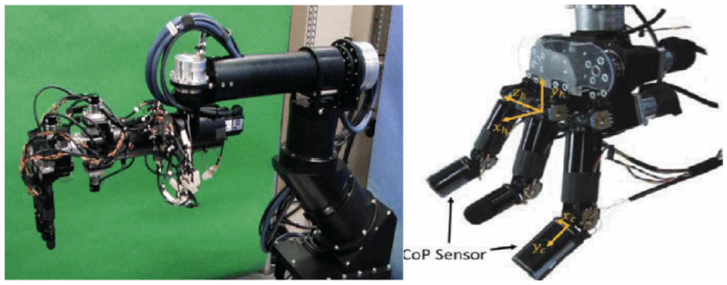


Figure 2.4: High-speed hand-arm juggling robot developed by Tomoki Oka, Naohiro Komura and Akio Namiki[23].

Also in 2012 Jens Kober, Matthew Glisson and Michael Mistry at Disney Research took playing catch with a robot to the extreme by adding two more balls, making it the first human-machine cooperative juggling robot. Figure 2.5 shows the robot *Sky*, a Walt Disney Imagineering A100 Audio-Animatronics, a hydraulic 37 DOF robot typically used in theme parks and its slightly modified end effector hand. ASUS Xtion PRO LIVE, a combined color and depth camera system, was used to track the balls by masking the color image with the depth image and using Hough circle detector. This vision system was running at 30 Hz and placed on the side, between the robot and the juggler [19].

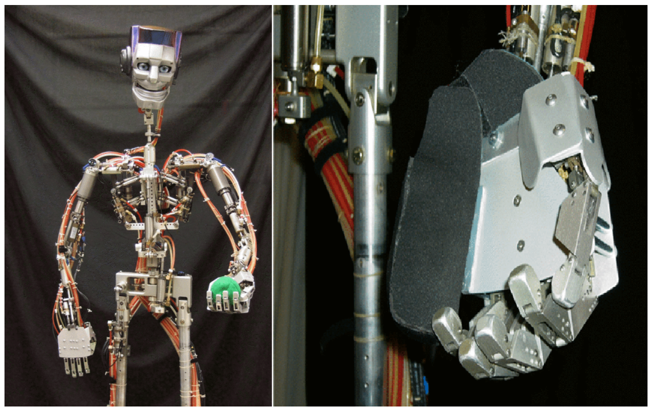


Figure 2.5: Disney robot *Sky* with its reinforced juggling hand to the right[19].

2.2 Vision System

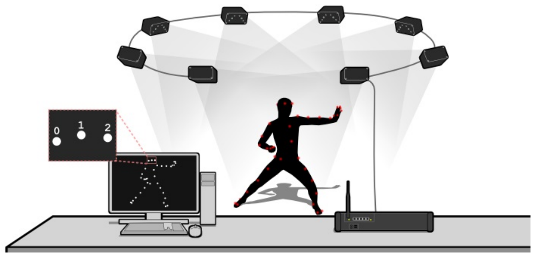
This section covers the vision system, which is necessary in order to achieve autonomous juggling. The purpose of this system is to continuously gather information about the location of the juggling balls through visual input. In this project the vision system is defined as everything observed by the cameras as input to the estimated landing position as the output. To achieve this, the vision system consists of cameras, a camera calibration software, a software that tracks the ball and calculates the depth of the ball, transforming pixel coordinates to real-world coordinates and a trajectory estimator that estimates the trajectory and landing position of the ball.

2.2.1 Marker Based Motion Capture

One of the most commonly used methods to detect object movements are *opto-electronic systems* based on *markers* (see Figure 2.6a). With the help of infrared light, 3D positions of objects can be detected by cameras. Moreover, markers can be either *passive* or *active*, meaning that the infrared light can be either reflected or emitted from the markers respectively. Usually, there is a need for several cameras in order to allow for a more precise object detection; however, this can contribute to a less mobile system [31].

Marker-based motion capture systems are considered to be the most accurate tracking solution, because markers are physically attached to the object that is going to be detected (see Figure 2.6b). Nevertheless, the drawback of using markers in motion analysis, is the fact that they are sensitive to everything that is being included in the capture environment. Sunlight for example, which includes an infrared component, can contribute with measurement noise. Other reflective surfaces, apart from the markers, can also affect the system negatively [31].

(a)



(b)

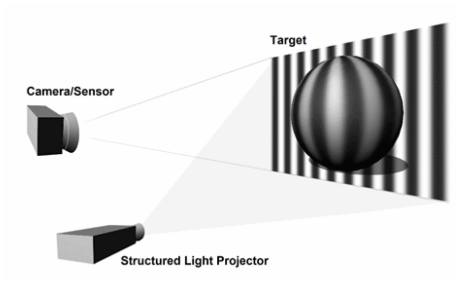
Figure 2.6: Marker-based motion capture systems converts real-time setups into a digital representation (b) with for example spherical markers (a).

One commonly used motion capture system is *Vicon*, which was the one used in the previous iteration of the JuRP project. It is a high-end motion capture system that is used both in research and entertainment. The system uses infrared cameras specially created for motion capture within a wide range of applications. The infrared light is both emitted and registered by the Vicon camera itself. These camera systems are usually very precise but they require extra preparation placing markers, and can be sensitive to reflective surfaces in the surrounding environment. The calibration of infrared cameras usually is done with a *calibration wand* which has markers fixated on it [18].

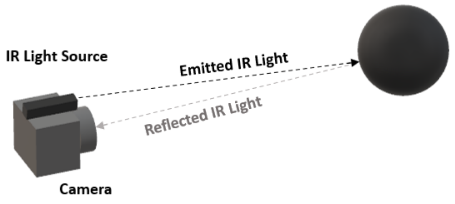
2.2.2 Markerless Motion Capture

As indicated by the name, *markerless motion capture* does not use any additional physical markers to capture or analyse the motion of an object. Instead these systems combine optical inputs with specialised algorithms.

Markerless systems can be divided into two different categories, *active* and *passive vision systems.* The active systems make use of emitted light from the spectrums of visible or infrared light. The depth in the environment can then be sensed either through deformations of a known projected pattern (*structured light*) or by measuring the time it takes for a pulse of light to return to the camera (*time-of-flight*), as seen in Figure 2.7. It is possible to get very accurate measurements when using active systems such as structured light systems and time-of-flight sensors, provided the measurements are static and collected in a controlled laboratory environment. This means that the active system alternative is not suitable when portability is a requirement, as it would limit the environments in which the system could be used [22].



(a)



(b)

Figure 2.7: Both the Structured light system (b) and the Time-of-Flight system (a) uses infrared light to measure the depth. (Image (a): [13])

Passive markerless systems only rely on information directly from the surrounding environment, captured in the form of images. The most common methods for gathering depth information in passive systems are *stereo* or *multi-view matching* where two or more RGB cameras are used instead of one to capture depth. These systems are usually the cheaper choice compared both to marker based systems and active markerless systems. The passive markerless system is not sensitive to sunlit environments and thus may be more suited for a mobile system than a marker based system. Though like any other system it has its flaws. The passive systems have experienced problems related to complicated scenes, occlusion and scenes containing areas with a high lack of texture [38].

Regarding markerless motion capture, one commonly used sensor is the *depth* or *RGB-D* camera. These cameras make use of either the Time-of-Flight or the Structured Light method, mentioned earlier in this chapter, for depth sensing. An RGB-D camera usually consists of three main parts, an IR projector, an RGB camera and an IR sensor. Together, these components can provide an image and depth information for all of its pixels. An example of a juggling system using depth cameras as vision is the Disney robot Sky, mentioned in the previous chapter, which used the ASUS Xtion PRO LIVE sensor. Benefits of the depth cameras are, among others, that there are commercial ones available at relatively low costs and they do not require preparations such as placing markers since they only rely on the environment itself. Though, compared to the marker based alternative the commercial depth cameras usually have a smaller field of view and a depth resolution that deteriorates notably as the depth increases.

When using a stereo setup for motion capture, common sensors are first of all stereo cameras. Consisting of at least two lenses, each with their own image sensor. The stereo cameras can simulate human binocular vision in order to achieve depth perception. The stereo systems does not perform well where there is a lack of details and texture. Alternatives to the stereo camera is using RGB or high speed cameras to create a stereo vision system. When taking this approach it is possible to create a system that fits the specific needs of the system as a whole. When fast moving objects are to be tracked the high speed cameras are a good choice. For more slow paced movements a regular RGB camera might be the better option since it would require less computational power. Moreover, the calibration process for RGB and high speed cameras are similar; it often involves taking multiple pictures of a chessboard pattern, getting intrinsic and extrinsic parameters that can later be used to avoid internal and external flaws in the cameras.

2.2.3 Object Detection

Before any depth information can be generated the object to be tracked needs to be detected. This is done with the help of algorithms that extracts the object of interest from the rest of the image. Usually the detection is done using a combination of different methods. Following are some of the more common methods which also are available .

Harris Corner Detection
This method is an improved version of the *Moravec's corner detector*. The idea in Moravecs version is that the changes in image intensity are determined by shifting a local window, within the image, in different directions. Doing this, three different cases that needs to be considered may occur. If the shift only results in slight changes of intensity, there is not a corner present. If instead the window reaches over an edge, large changes will occur when shifting the window in a direction perpendicular to the edge, but the changes will be small when shifting along the edge. The third case considers the possibility of the patch within the window being a corner or an isolated point, then the changes in intensity will be large for all shifts. The similarity of intensity between two patches is determined by calculating the sum of squared differences between the two patches corresponding pixels. In Harris version this calculated value is related directly to direction rather than a patch of pixels [6].

Background Subtraction
A segmentation method used to isolate moving objects in a video stream by segmenting the images into background and foreground. The background usually acts as a static reference image and the foreground contains the moving objects.

Circle Hough Transform
A feature extraction method that focuses on finding circular shapes in an image. Using this method, an edge map is calculated. This edge map is then used to find edge points that will map 3D space into a conical surface. By then looking for points where at least three different cones intersect, parameters of a circle can be found [37].

Color Isolation with HSV
Objects can also be extracted after their *hue*, *saturation* and *value* (HSV) levels from an image. The algorithm includes steps such as converting an image's RGB color space to HSV because it is easier to isolate colors with HSV. After this conversion, the images are to be masked. There are two types of masks that needs to be applied to the frame when color isolation is used. The first mask includes setting a lower bound of HSV-levels while the second mask includes the setting of an upper bound instead. Parts of the image which are within these threshold values will then be able to be extracted from the frame [28].

2.2.4 Depth Calculation

By having the cameras in stereo setup, triangulation can be performed and depth to the ball can be calculated using the theory below.

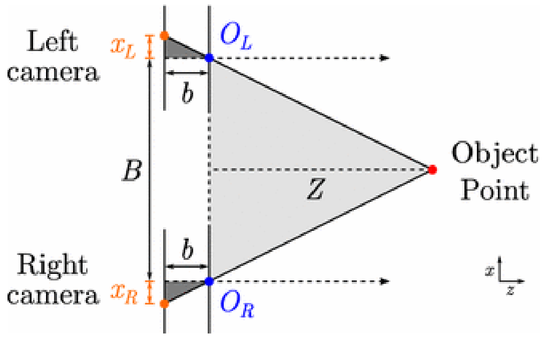


Figure 2.8: The distance Z between the cameras and the object point can be found with triangulation [5].

By using the disparity, d, between where the object appears on each lens and knowing the distance between the cameras, B, and the focal length, f (in pixels), the depth of the object can be calculated, see Figure 2.8 and the equations below.

$$d = x_L - x_R = f\left(\frac{X+l}{Z} - \frac{X-l}{Z}\right) = \frac{2fl}{Z} \tag{2.1}$$

$$Z = \frac{2fl}{d} = \frac{Bf}{d} \tag{2.2}$$

As the digital image is discrete in its nature, the disparity values will be integers unless special algorithms are used to compute disparities on a sub pixel accuracy. To increase the accuracy of the calculations, the distance between the cameras, B, can be increased. This will increase the disparity and will reduce the negative effect of the discrete nature of the pixels. This will however decrease the view area which is seen by both cameras [8].

2.2.5 Coordinates Transform

The output from the vision system needs to be a position in real-world X, Y, Z coordinates for the inverse kinematics solver to work. The camera and the tracking algorithm get the ball's position in pixel coordinates from the camera coordinate system, see Figure 2.9. Therefore, a coordinates transform to the world coordinate system, illustrated in Figure 2.9, is needed for the inverse kinematics solver to function correctly [8]. The Z-coordinate attained from the triangulation will already be in the real-world unit. In order to get the corresponding X and Y coordinate these equations can be used:

$$X = \frac{x_L Z}{f} \tag{2.3}$$

$$Y = \frac{y_L Z}{f}, \tag{2.4}$$

where x_L and y_L are the corresponding x and y pixel-coordinates from the left camera and f is the focal length in pixels.

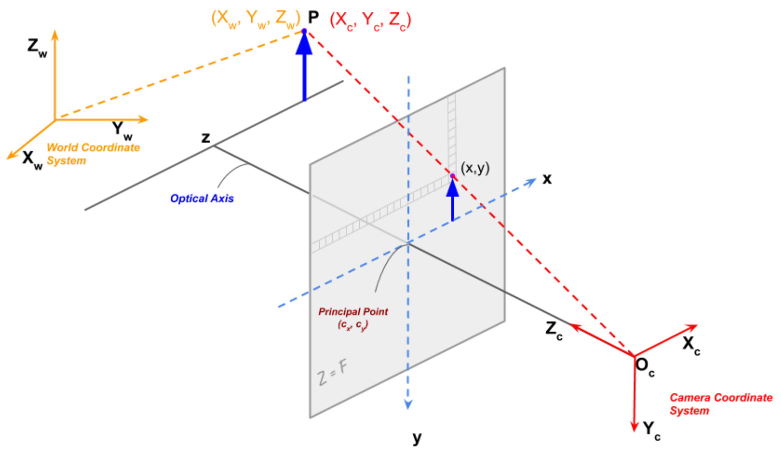


Figure 2.9: Coordinate transform from a camera coordinate system to the world coordinate system. P represents a point in the real world.

2.2.6 Trajectory Estimation

A trajectory estimator estimates the landing position of the ball while it is in the air. With a trajectory estimator, the robot can position the end effector in the estimated landing position before the ball reaches it. By knowing the current and all previous positions of the ball, an estimation of future trajectory can be done and sent to the computer from the vision system. First, calculate the ball's velocity by comparing the time difference between the change of position in frames. Secondly, knowing the acting gravitational force and the velocity, the trajectory can easily be solved using simple physics. It is implemented in either three or two dimensions, with two dimensions being the most common.

MATLAB's Built-in Trajectory Estimator
MATLAB has a built-in function for trajectory estimation which makes coding a program redundant, however, MATLAB will be needed as a node in the system.

Euler's Method
Euler's method is a first-order numerical procedure for solving ordinary differential equations. This method applied to the trajectory and modelled with gravity as the only external force is a common way to implement a trajectory planner. There are many different scripts of this implementation on GitHub. To get a good and constant output, the input needs to have as little noise as possible. Because of the method only taking in the two previous positions, the algorithm is prone to flickering and uncertainties when introduced to noise. This requires a low-pass filter or other countermeasures [9].

Kalman Filter
A kalman filter is a filter/algorithm that predicts future data based on previous and incomplete or noisy data. The prediction improves with larger data and time. It has many different implementations in many different fields and is more complex than Euler's method which is the reason why it is generally slower [16].

2.3 Software (excluding vision system software)

It requires a great range of actuators, sensors, processors, and other hardware for the robotic arm to be able to juggle. All these devices contain software that is handling all the logic and calculations. There are a lot of different software needed in projects such as robotics juggling.

Below are some of the most commonly used:

- An *inverse kinematics solver* which is given the input of a position (the estimated landing position) and outputs the necessary angle of the motor making the end effector to be positioned at the estimated landing position.
- A *low level controller* optimizing the speed and angles of the motors to be as quick and precise as possible.
- A *Robotics Middleware* software communicating with the different programs as well as with other hardware as computer and micro-controllers.

In addition to these, other types of software are helpful to use during development such as:

- A *simulation environment* where the JuRP arm can be tested.
- A *CAD* software where the design of the arm can be developed and parts can be 3D-printed to be used as physical parts. Weight and center of mass can also be easily calculated before construction as well as used as a model in the simulation environment.
- A *software for modeling, simulating and analyzing dynamical systems* for calculating correct force and speed of the joints, motors and other parts.

2.3.1 Inverse Kinematics Solver

To have a fast inverse kinematics solver (IK solver) it is crucial for the system to have a low computation time, because to solve the inverse kinematics can be very time consuming.

IKfast	IKfast is a powerful and fast inverse kinematics solver which can run as fast as 5 microseconds and it can generate language-specific files.
TRAC-IK	TRAC-IK is an inverse kinematics solver that combines two inverse kinematics implementations via threading to achieve a more reliable solution than common IK solvers.
KDL	KDL inverse kinematics solver is the default IK solver in ROS. It is not as fast as other IK solvers.
Matlab IK	Matlab has a fast inverse kinematics solver which is easy to use. However it will need Matlab as a node in the robotic network which will slow down the system.
Geometric approach	for simpler systems, the inverse kinematics can be solved geometrically. Firstly, setting up an equation system from the forward kinematics. Secondly, solving for the angles.
Numerical approach	The angles can also be solved numerically by using the forward kinematics matrix and the transformation matrix's jacobian.

2.3.2 Robotics Middleware

There are a wide variety of robotics middleware available with none dominating. The purpose to have a robotics middleware is to have all the different computers and devices communicating with each other as well as being able to simplify software design and hide the complexity of low-level communication.

ROS	ROS is a popular, fast, open-source robotics middleware run on the Linux operating system. It is used in many similar projects and is popular in the academic world with a large community and many tutorials on how to use it.
Player Project	Player Project is used to create free software for research into sensor systems and robotics. It is one of the most popular open-source robot interfaces in research and academia.
Urbi	Urbi is an open-source software platform in C++ used to develop applications for complex systems and robots. Urbi is based on the UObject distributed C++ component architecture
ROS2	ROS2 is a popular, fast, open-source robotics middleware run on the Linux operating system. It is an successor of ROS with new features and Real-time support.

2.3.3 Robot Simulation Environment

Robot simulation environment or a robotics simulator is a simulator used to test and simulate the robot without the need for the physical machine.

Gazebo	Gazebo is a popular open-source 3D robotics simulator which has been used in many large robotics competitions including multiple DARPA robotics challenges, Toyota Prius Challenge and NASA space robotics challenge. It can import CAD-models as URDF-files into the simulation.
SimSpark	SimSpark is an open-source robotics simulator commonly used in academic research and education in the fields of AI and robotics research.
Webots	Webots is open-source 3D robot simulator used in the industry, academia and research. It was originally proprietary software but was released as open-source in 2018. It includes a large collection of models of robots, actuators and objects and also have the ability to designing a model from scratch.

2.3.4 Computer-aided Design

To design the arm a CAD software is used to design all parts. These CAD-parts can then be 3D printed to be used as parts on the physical arm as well as used in simulation.

SolidWorks	SolidWorks is a CAD software that is proprietary and is not included in KTH's program distribution library. However, it can export the part to URDF-file format which is used in most robot simulation software.
Solid Edge	Solid Edge is a CAD software that is proprietary and is included in KTH's program distribution library but it cannot export models to URDF-file.

2.3.5 Physics Simulation

To calculate the forces, speeds and weights needed from the motors and length, thickness and weight of the arm, physics simulation and calculation software is needed.

Matlab	Matlab is a proprietary software program with its own high-level programming language intended for numerical computations. It is very popular in engineering and science. It is provided in KTH's program library distribution.
GNU Octave	GNU Octave is a software program with its own high-level programming language intended for numerical computations. The language is intended to be as similar to Matlab's as possible to be as bi directional compatible as possible. It is one of the major free alternatives to Matlab.
Mathematica	Mathematica is a popular proprietary technical computing system. It is not provided in KTH's program library distribution.

2.3.6 Computer Hardware

More computers and hardware means more computation power and fast computation time. However, the time it takes for the different hardware and computers to communicate with each other slows down the system.

2.4 Communication

There are many different ways to build up a communication network for a robotic system. This communication network has to be stable, robust, and fast in order to send the signal without delays or getting messages lost.

There are different architectures of communication. One Architecture could be like a human brain, where the left part controls the right part of the body, and the right part of the brain controls the left part of the body. Each part has a role to play in the processing of information although the other is more dominant in certain functions or purposes [33]. In order to choose a communication method, more understanding of different communication ways is needed.

2.4.1 USB

Universal serial bus (USB) is an industry-standard that establishes protocols for communication and power supply between computers and other devices. Examples of peripherals that are connected via USB include computers, microcontrollers, and cameras. In standard USB 2, data can be transmitted at a speed of up to 480 Mbit/s.

USB cables are limited in length, as the standard was intended for peripherals, not for long distances between rooms. USB also has a strict tree network topology, which is ilustrated in Figure 2.10. Devices connected with USB can not interact except via the host, and two hosts can not directly communicate over the USB ports. A Universal Serial Bus port receives a serial bit stream of data and commands in a Universal Serial Bus protocol from a USB host computer [11]. All peripherals connected to the host must be addressed individually i.e. a host can not broadcast signals to all of them at once.

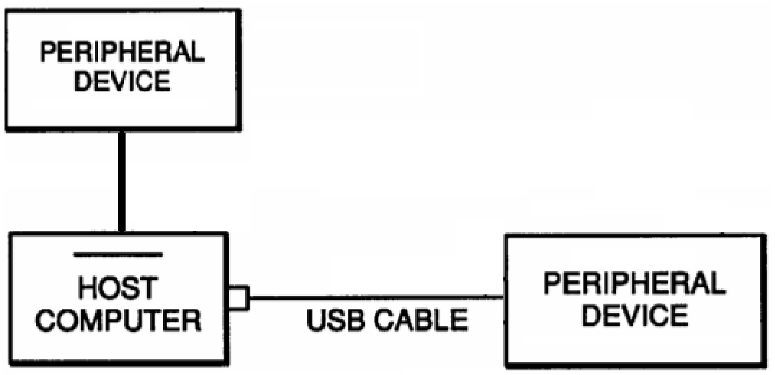
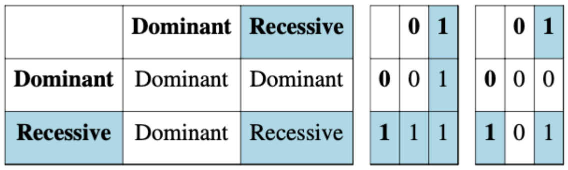


Figure 2.10: Networks with one host and two devices connected with USB.

2.4.2 CAN

Controller Area Network (CAN) is a message-based protocol and was initially developed for the automotive industry for a safer and more robust handling of the increasing amount of electronics in the vehicles. This broadcast communication makes it possible to have a low-cost and lightweight network and allows devices to communicate with each other without a host computer [20]. Devices are divided into nodes where each node can both send and receive messages between each other, but not simultaneously. Each device can decide to either receive or filter the message (see Figure 2.14). The messages are prioritized to avoid collisions, the priority is set by its ID (identifier). Logical zeros are the dominant bits on a can bus and whatever message with the most dominant bits in their ID will momentarily overwrite less dominant messages (see truth table Figure 2.11) that are sent simultaneously on the bus [20].



	Dominant	**Recessive**
Dominant	Dominant	Dominant
Recessive	Dominant	Recessive

	0	**1**
0	0	1
1	1	1

	0	**1**
0	0	0
1	0	1

Figure 2.11: The truth table for prioritising IDs [20].

The physical layer bus consists of two twisted wires (CAN high & CAN low) terminated with two 120ohm resistors in both ends of Figure 2.12. The only limit to the number of devices able to run on the same CAN bus is the number of possible message-IDs, but there must be at least two devices connected to the bus for transmitting and receiving signals.

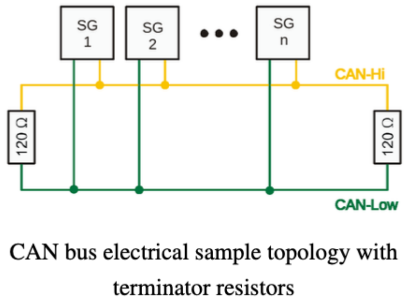

SG 1
SG 2
SG n
CAN-Hi
120 Ω
120 Ω
CAN-Low
CAN bus electrical sample topology with terminator resistors

Figure 2.12: CAN bus physical topology example with terminator resistors [20].

The CAN bus consists of a differential signal, meaning that it communicates using two mirrored signals, CAN high [2.5V, 5V] and CAN low [0.0V, 2.5V], see Figure 2.13. The bus is at a recessive state (logical 1) when both CAN high and CAN low transmits 2.5V. The CAN bus asserts a dominant state (logical 0) when the CAN high goes to 5V and the CAN low goes to 0V.

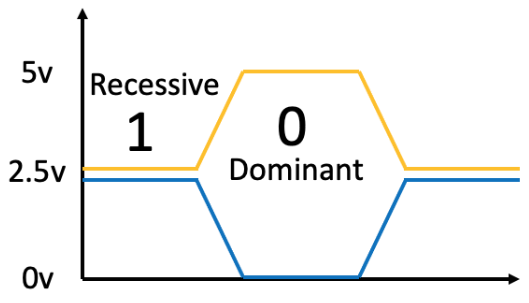


Figure 2.13: Illustration of the signal logic in can protocol.

A High-speed CAN network allows communication at transfer rates up to 1 Mbit/s which gives enormous benefits in robotics where the signals have to be transferred fast to actuators.

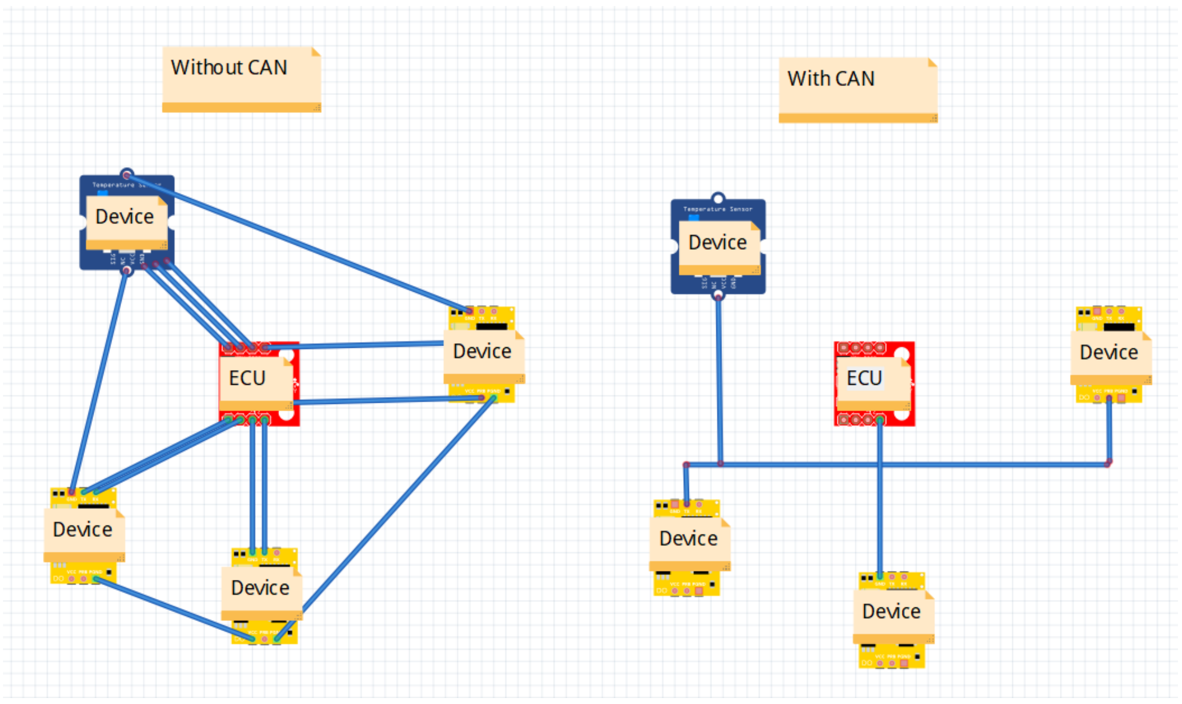


Figure 2.14: Networks with vs without CAN.

2.4.3 Ethernet

EtherCAT (Ethernet for Control Automation Technology) is an Ethernet-based system, that is suitable for real-time computing requirements in automation technology. The machine structure determines the network topology in EtherCat. One advantage of EtherCat is that EtherCAT doesn't need hubs or switches. It makes it possible to have a line, tree, star topologies, or any combinations with an unlimited number of nodes.

In EtherCAT, every datagram contains the info that the receiver needs to grab the datagram intended for it, and then to reassemble them perfectly into each unique piece of message that was sent to the device. So when the receiver opens your message, the datagram is perfectly reassembled, and they look just the way they did when they were transmitted.

2.5 Robotic Arms

The mechanical design of most robot arms are based on the humanoid arm. A shoulder part, a transverse part, an elbow, and a wrist will give 6 degrees of freedom. One example is from the manufacturer *ABB*. The model is called *IRB120* and is quite stable and robust, seen in Figure 2.15.

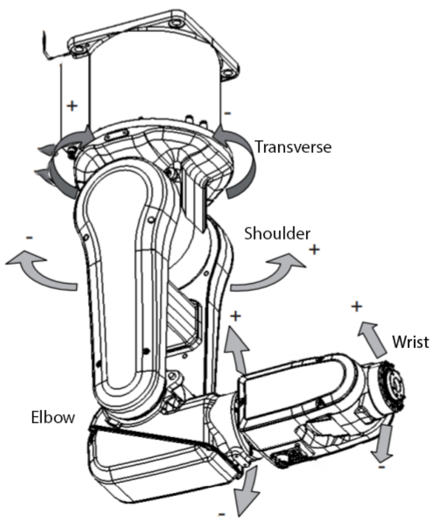


Figure 2.15: *ABB IRB120* with respective joints [1]

A different version of arms used for robotic juggling is shown in Figure 2.16. These mechanical arms are not shaped like humanoid arms. Instead they are solid metal beams mounted to a rail which moves up and down. The arms can also travel in a transverse direction, allowing for a throwing motion. It is proven to be an efficient design for a certain type of juggling.

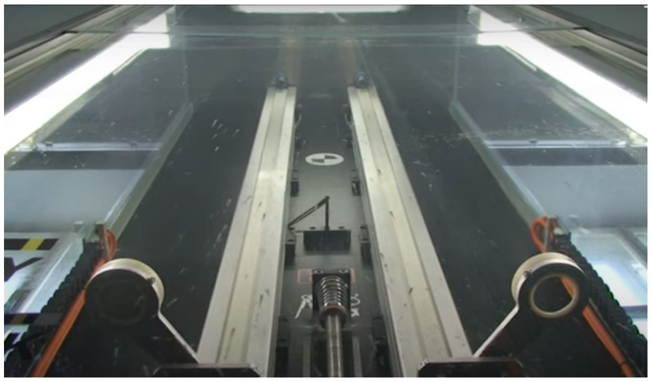


Figure 2.16: Example of juggling robot from Department of Control Engineering at Prague's Czech Technical University [10].

2.6 End Effector

End-effectors plays a vital role in the arm of a juggling robot. They are designed differently depending on certain requirements such as shape of the gripping object, the precision need and forces involved. Because of the end-effector's placement on the most exterior part of the arm, light weighted ones are preferred to avoid high stress concentrations. Furthermore, vibrations in an end-effector are undesired due to its negative effects on the positioning accuracy [38]. By taking these factors into consideration, many juggling robot projects consider *cones* and *grippers* to be most suitable for ball manipulation (see Figure 2.17).

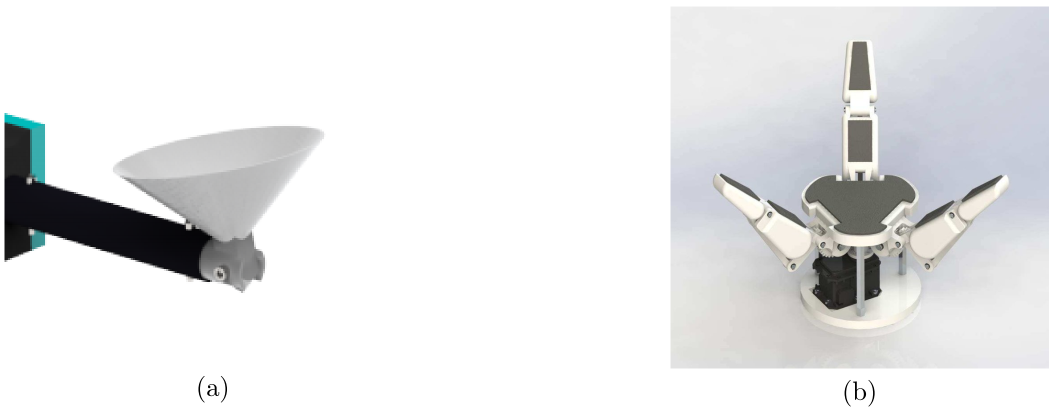
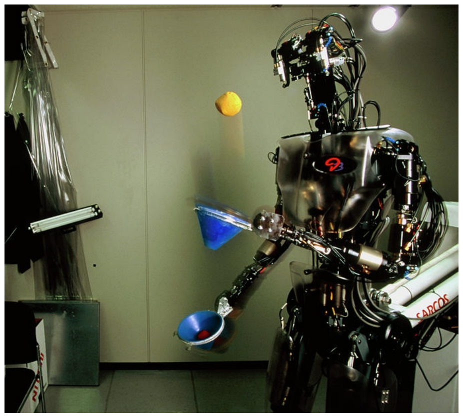


Figure 2.17: The most commonly used end-effectors in juggling robots; cones (a) and grippers (b).

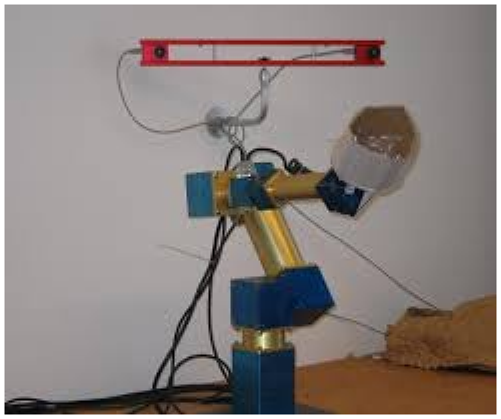
2.6.1 Cones

Funnel shaped end-effectors (see Figure 2.17a) are known for being suitable for ball catching [19]. Moreover, cones are both inexpensive and simple in its design, making this option a definite alternative for many juggling robots. Because there is no clutching involved, this design is not in need of any additional motors, making cones one of the more light-weighted alternatives. However, the lack of gripping ability, increases the risk of the ball bouncing out of the end-effector during catching, which has been a common problem in many similar projects [30]. Another consequence of having no clutching function, is the fact that the throwing and catching becomes less precise and stable.

An example of a juggling robot that uses funnel shaped end-effectors is the *Sarcoman* juggling robot made by the company *Sarcos Robotics* in 1995, see Figure 2.18a. In the same figure, another project done by Christian Smith, Mattias Bratt and Henrik I Christensen at the *Royal Institute of Technology* in 2015 can be seen as well. The last mentioned example, is a ball catching robot that uses teleoperation.



(a)



(b)

Figure 2.18: Examples of two different projects, Sarcoman (a) and the teleoperating robot (b), were cups have been implemented as end- effectors.

2.6.2 Grippers

Another end-effector type that is widely used in robotics, especially in ball catching and throwing projects, is the three-fingered gripper, see Figure 2.17b. The three fingers resemble the shape of a funnel, which is why this alternative is found to be suitable for manipulating balls. Furthermore, the precision of grippers are higher compared to other types of end-effectors. Grippers can also be made at a low cost and light weight if they are for example 3D-printed. Otherwise, these end-effectors are known for being one of the more expensive ones on the market [35]. The consequences of using grippers in a juggling robot is the complex design, making implementation more difficult; the more fingers that are attached, the greater the implementation complexity the gripper will have. The project of the baseball catching robot by Masatoshi Ishikawa from *Tokyo University* is an example of a 2009 study that used a three fingered gripper, see Figure 2.19.

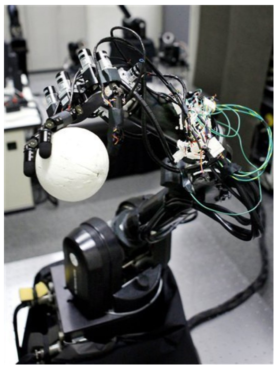


Figure 2.19: Masatoshi Ishikawa developed a baseball catching robot in 2009 at Tokyo University.

Chapter 3

Methodology

3.1 Project Management

An agile project management style was used, specifically a mixture between Rapid Results and Scrum. "Rapid Results" were invented by *Schaffer Consulting* and is a structured process that mobilizes smaller teams to achieve dramatic results, formed under the pressure of short time frames and ambitious targets. By targeting smaller versions of the finished product earlier in the project, "Lessons Learned" and integration problems are detected before it's too late. From the management method Scrum, "runs" were adapted with the duration of one week. At the beginning of every week, the team members briefed each other on the week's tasks and organized them in a digital *Kanban board*.

3.2 Design Methods

Literature studies and interviews with industry experts acted as a basis for component choices. There were a substantial amount of simulations and calculations done by the preceding groups, concerning the power and force requirements of the existing design. Thus, this was used as a basis for the extension of the adequate components and conceptual design choices. Interviews with industry experts from ABB's robotics lab lead to an update of the old communication system, bringing it to industry standard. The small budget enforced a trade-off between system robustness and available materials. The group's manufacturing knowledge shaped the design in parts that couldn't be ordered due to time and/or budget limitations.

3.3 The V-Model

Many of the developments in the project was done according to the V-model. Meaning the problem was first formulated as a grand concept, then divided into several smaller parts in detail. Each of the smaller parts was then evaluated and when approved incorporated into the system at large. This meant that each smaller part was continuously tested, and when the entire system was assembled most parts functioned as intended with little issues.

Chapter 4

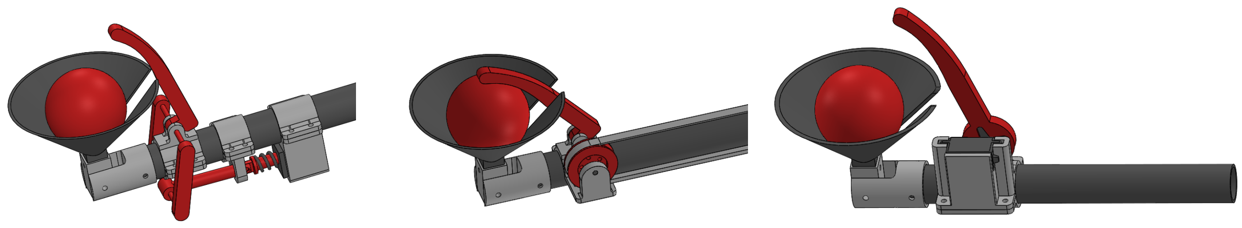
Concept Design

In the process of redesigning the old prototype, to better fit the requirements specified for this years goals, several alternative concepts were created. This chapter will present these alternatives in order to provide a background to some of the final choices made in the next chapter.

4.1 End Effectors

When designing the end effector a few factors were considered. First of all, the new design needed to achieve more precise catching and throwing without the risk of the ball falling out. Due to the placement of the end effector, its weight was also an important design factor. A heavy end effector puts unnecessary stress on the motors moving the arms. Thus, the weight needed to be kept at a minimum. Based on these factors, two main concepts, each consisting of a few sub-concepts were created.

The first concept built upon the previous end effector using a cone, with the addition of a lever that locks the ball in place between throws. It consists of three sub-concepts, that can be seen in Figure 4.1 below, each making use of different actuators and mechanisms to keep the ball in place.



(a) (b) (c)

Figure 4.1: The three alternatives of cone and lever concept that represent the use of different actuators and mechanisms.

Alternative *a)* uses a solenoid in combination with a linkage mechanism. This solution has a low computational complexity since the solenoid can be controlled by just providing it with a current. Though, with

both the solenoid and the rest of the components placed on the robot forearm a lot of weight is added. The mechanical complexity is also higher than that of alternative b) and c). The high number of moving parts increase the risk of parts getting stuck due to wear.
Alternative *b)* uses a timing belt to rotate the lever with the help of a motor placed in the elbow. Depending on the choice of motor the computational complexity can be at different levels, but higher than that of alternative a). With the motor placed in the elbow, the weight will be better distributed. However, the remaining components will still add to the total weight of the end effector. This alternative also has a lower mechanical complexity than alternative a).
The above-mentioned alternatives were at first the only two. However, due to the importance of a lightweight end effector, and the fact that these alternatives added relatively much weight, there was a need to find something lighter. This led to **alternative *c)***, in which a small servo would be used. Due to its light weight (many being less than 80g) it could then be placed right next to the end effector, meaning the only added weight apart from the cone and lever would be the motor and a mount, making this a relatively lightweight option. This alternative also has the lowest mechanical complexity of all the alternatives. The computational complexity is higher than that of alternative a) but still quite low since the servo has built in feedback control.

The second concept takes inspiration from existing 3-finger grippers within robotics. Instead of a cone this concept makes use of several levers moving simultaneously in order to close around the object, locking it in place. The idea was to make the levers move using a combined linkage and wire mechanism with a motor in the elbow joint. When the motor rotates it pulls a wire that drags a plate connected to the levers down, resulting in the gripper closing. The two alternatives seen in Figure 4.2 below are mostly the same. In this case the difference only lies in the shape of the fingers.

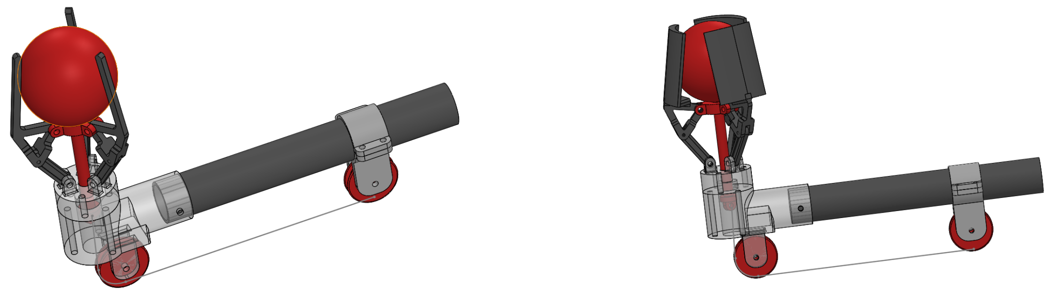

(a) (b)

Figure 4.2: The two gripper concepts are the same in function but alternative b) has wider fingers to better keep the ball in place, but this also leads to a higher weight.

This concept could allow for a more human like catch and throw and more control over locking and letting go of the ball. Problems may be that it is one of the more, if not the most, mechanically complex concept. It is also the heaviest option even with the motor at the elbow, since the mechanism itself adds a lot of weight.

The different concepts were evaluated against each other with the help of a decision matrix, as seen in Table 4.1. In this matrix each concept is given scores on a scale of 1-100, with 100 meaning that the concept is optimal, based on how well they manage important design factors. These factors are given a value depending on their level of importance. The higher the value, the higher impact this factor has on the functionality of the end effector.

Table 4.1: Weighted decision matrix used to evaluate the different end effector concepts.

	Non-obstructive	Weight	Complexity - Mechanics	Total
Weight	0.5	0.5	0.2	
Cone (Solenoid)	90	15	40	60,5
Cone (Belt)	90	50	50	80
Gripper (Wide)	85	45	20	69
Gripper (Thin)	85	40	20	66,5
Cone (Servo)	90	80	85	102

The total score of a specific concept is then given by the total sum of each of the factor values/weights multiplied with the concept score for that factor. The concept with the highest total score should be the one that manages the design factors best, and the one to continue working with. As can be seen in Table 4.1, the cone concept using a servo got the highest score and thus became the one used in this project. A more detailed explanation of the chosen concept can be found in chapter 5.

4.2 Motors

The choice of motors was the subject of a great deal of discussion, especially at the early stages of the project.

- The *Shoulder* motors at the top of the arm needs to provide a great deal of torque to move the entire weight of the arm.
- The *Transverse* motors need to be able to rotate the arm at an adequate speed, and provide good design opportunities to facilitate good cable management and proper weight distribution.
- The *Elbow* motors need to be light, yet provide great speed, as they are the main source of movement for the throws.
- The *End effector* motors needs to be very light, have a quick response and provide a binary movement pattern, *open* or *closed*.

The three first motors also need to provide feedback, as the eventual goal is position control. These requirements make the choosing of suitable motors for each location a non trivial part of the project.

In Table 4.2 some generalized advantages and disadvantages with different types of motors are presented. Not that these are not applicable for ALL motors, but can be used as a general baseline.

Table 4.2: Advantages and disadvantages of different types of motors.

	Brushed	**Brushless**	**Servo**
+	· Cheap · Easy to control	· Efficient	· Encoder built in · Easy to control
-	· Need external Encoder · Low Efficiency	· Expensive · Need specialized drivers to control	· Expensive

These different attributes show that if cost was of no concern, the preferable choice would most likely be to use integrated servo motors on all joints of the robot. However as the budget is limited, this was not an option.

The previous group, JuRP2019 produced some specifications for the different joints, shown in Table 4.3 below:

Table 4.3: Motor specifications.

Joint	Required Max Torque	Required Max RPM
Shoulder	9 Nm	55 RPM
Transverse	2 Nm	33 RPM
Elbow	9 Nm	128 RPM

These values were confirmed by calculations, and used as a helpful guide when choosing motors for each joint.

4.3 Motion Planner

4.3.1 Inverse Kinematics and Simulation Packages

Based on the literature review and last year's project recommendation, IKfast and thereby Moveit and ROS was the initial proposed solution to handle the inverse kinematics and motion planning of the arms. The main factor taken into consideration here was the execution time for the inverse kinematics solver. To realize this, a completely new framework and ROS workspace was needed. Last year's Universal Robot Description File (URDF) was modified to suit this year's upgrade. Two arms instead of one were used in the simulations of the inverse kinematics. The end effectors were disregarded in the URDF, considering its nonvital functionality to the rest of the arms' inverse kinematics solver and motion planning algorithms. For simulation, Gazebo was used with ROS to try and to test different movement patterns and control strategies. Figure 4.3 shows the arms in simulation. Note that the arms were mounted on last year's rig, as the design of the rig does not matter due to collision avoidance not being used in the simulations.

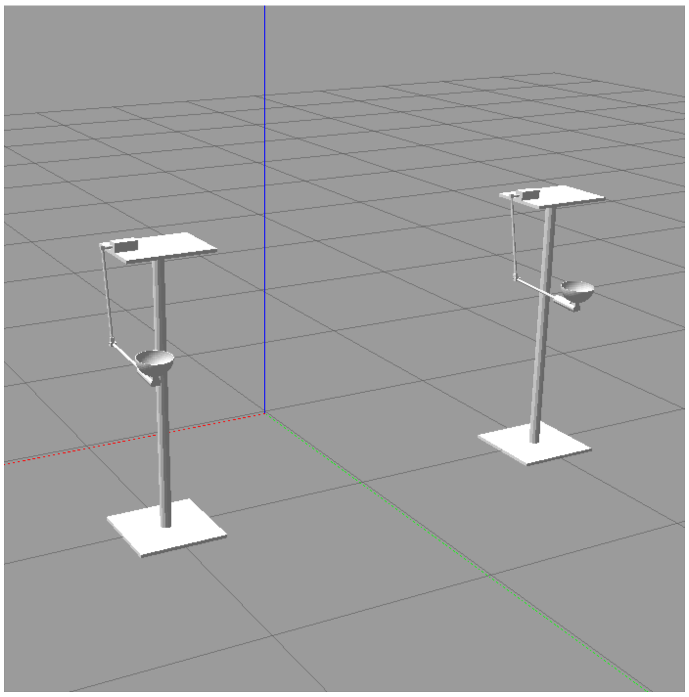


Figure 4.3: Model in Gazebo used for kinematics solutions.

ROS Moveit could handled the motion planning in both the simulations and real life movement. The software set up the necessary data flows and called the inverse kinematics solver to create a trajectory for the end effector from start position to goal position with the desired velocity. A ROS control hardware interface was realized to enable the same scripts in simulation to be used for the real robot. ROS showed to be a very useful middleware to set up the framework for all data flows and to structure the communication between nodes.

4.3.2 Numerical IK-Solver

A numerical inverse kinematics solver was also investigated. First, setting the frame coordinates according to the Denavit-Hartenberg convention (see Figure 4.4). Secondly, transforming the coordinates between the base-frame and the end effector (see Appendix A.3) creating the transformation matrix.

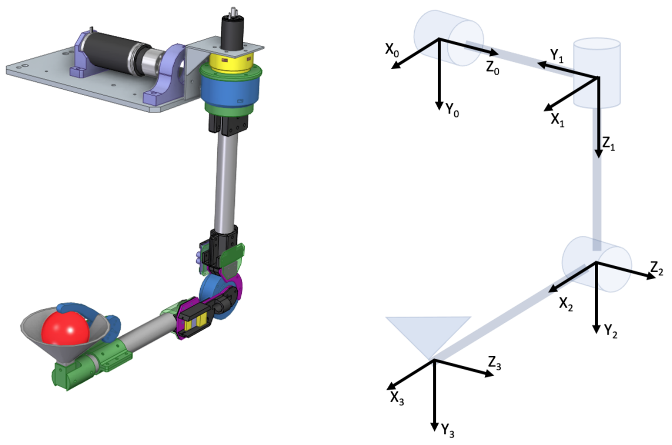
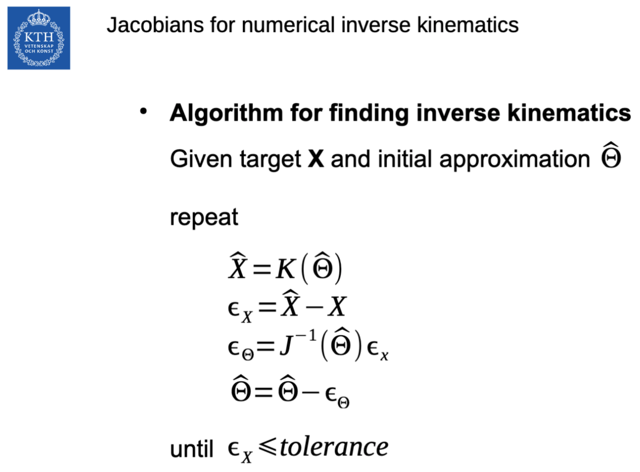


Figure 4.4: The joint frames according to the Denavit-Hartenberg convention.

The forward kinematics and jacobian could be extracted from the transformation matrix. The transformation matrix contains the forward kinematics for the end effectors position (D-part) in the top right corner. The forward kinematics, together with the transformation matrix's jacobian could be used to solve the inverse kinematics.
The pseudo-code (Figure 4.5) worked as a frame for numerically solving the inverse kinematics. The script received waypoints from the path planner, and used current position as an initial guess before traversing towards an answer within the user's required tolerance. For this approach to work, a good path planner must be built. The planner must give a sufficient, but not overflowing amount of waypoints to the solver. There must also be a sophisticated world frame system in place for both arms and cameras to sufficiently locate each other[21].



Jacobians for numerical inverse kinematics

- **Algorithm for finding inverse kinematics**

  Given target **X** and initial approximation $\hat{\Theta}$

```
repeat
    $\hat{X} = K(\hat{\Theta})$
    $\epsilon_X = \hat{X} - X$
    $\epsilon_\Theta = J^{-1}(\hat{\Theta})\epsilon_x$
    $\hat{\Theta} = \hat{\Theta} - \epsilon_\Theta$
until $\epsilon_X \leqslant tolerance$
```

Figure 4.5: pseudo-code for numerical inverse kinematic solver. Where K is the forward kinematics, $\hat{X}$ is the position estimate, ϵ_x is the position error, J is the Jacobian and ϵ_θ is the motor angle error[21].

Chapter 5

Implementation

In this chapter, all the final decisions will be presented. Giving details and a short reasoning why a certain choice was made.

5.1 Construction Design

As the project this year aimed towards creating a design which incorporated two arm juggling, many new design choices needed to be made, and a great deal of construction was required.

The robot was divided into several mechanical subsystems, *Rig*, *Transverse*, *Shoulder*, *Elbow* and *End Effector*, mainly based around a certain purpose or due to contributing to additional movement in a direction. The fully assembled system can be seen in Figure 5.1.

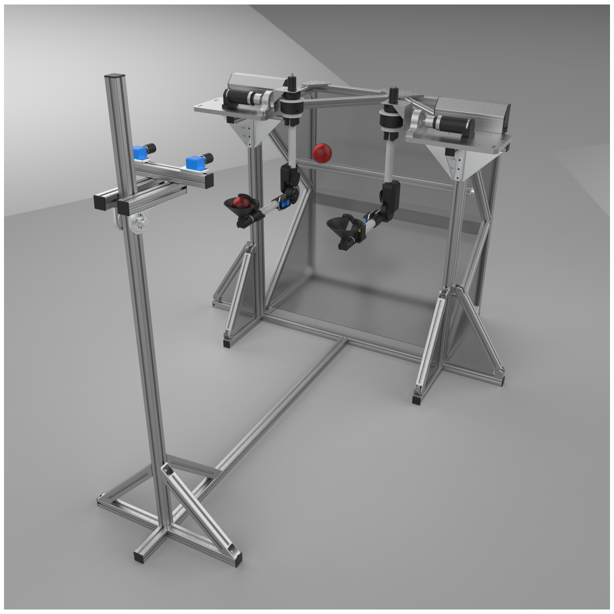
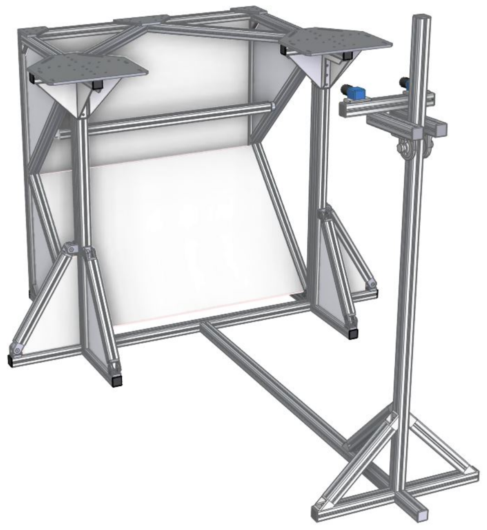


Figure 5.1: The full design of the system shown in CAD.

5.1.1 Rig

The purpose of the rig was to provide a stable and reliable base for the mounting of the arms. The rig needed to be robust and sturdy, yet provide adaptability for new choices and ideas to be implemented in the future. The design was first implemented in CAD, and then assembled in real life, as can be seen in 5.2 below.


(a)

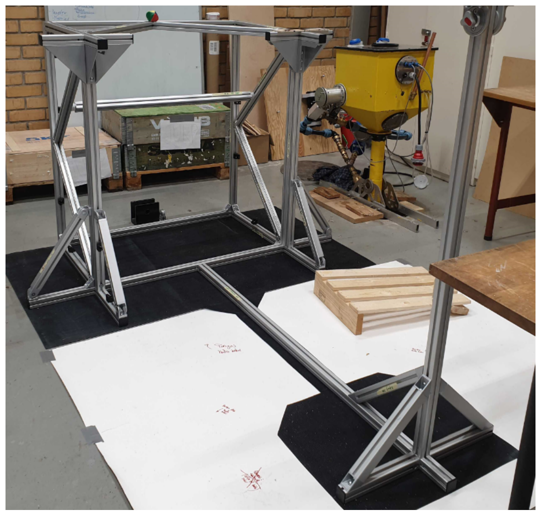

(b)

Figure 5.2: Side by side comparison between the CAD design and the real life implementation of the rig.

The rig was constructed using aluminum profiles. Because of this, once the material had been received, cutting and assembling could be completed in a relatively short time. The rig has supporting beams in several places, allowing to brace forces in many different directions, without extensive vibrations or movement.

For placing the arms, two 10 mm steel plates, with a number of holes were cut using a waterjet. The plates were then mounted on the front of the rig, allowing for a number of different mounting possibilities for different motor and support combinations. Metal boxes were also mounted on these plates, whose purpose was to contain the necessary electronics, protecting them from possible damage and facilitating a robust cable management.

The camera mounting was connected via an additional aluminum profile, which extended along the ground in front of the rig. In later experimentation, the distance was deemed too short and thus was moved further away from the main rig. Due to the design, the camera mount could easily be detached from the rest of the rig and moved backward.

5.1.2 Transverse

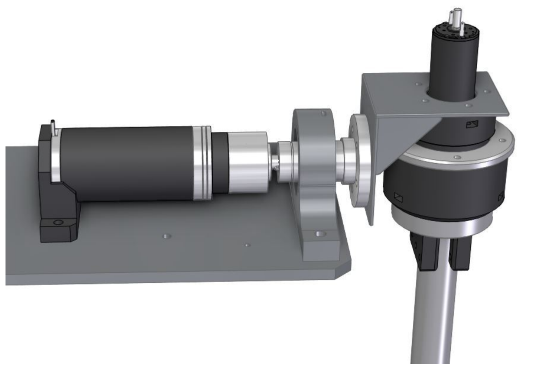
The transverse design on both arms was kept almost the same as it had been the previous year. Only some minor measurements were changed, in order to better fit the diameter of a different gearbox on the transverse motor *Faulhaber 3863CR*, and to reduce backlash in the plastic components.

5.1.3 Shoulder

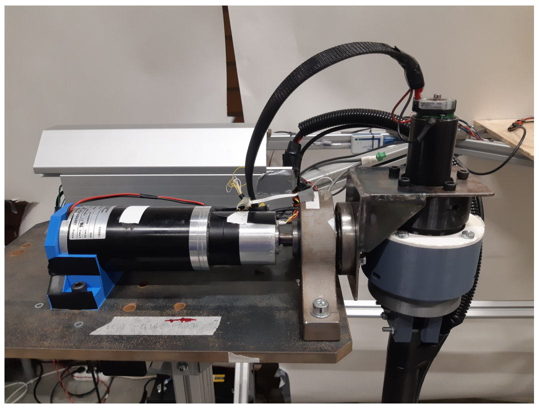
The shoulder was the place on the arm that needed to withstand the most amount of weight, and thus provide the most amount of torque. Thankfully it was also the most robust place due to the support from the rig and thus a heavy solutions could be implemented.

Arm 1

On the inherited arm, the design was kept very similar to how it was made the previous year. Only minor replacements and improvements were made. The planned design and the real life implementation can be seen in Figure 5.3 below.



(a)

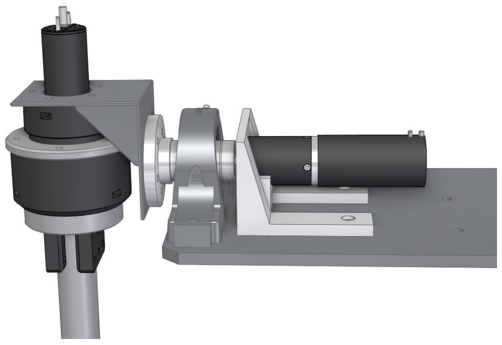


(b)

Figure 5.3: Side by side comparison between the CAD design and the real life implementation of the shoulder for arm 1.

The design uses a custom made bearing housing to lock the *Crouzet 8989B1* motor axis in place, and facilitate easy rotation in the shoulder direction. The back of the motor is fastened on a 3D-printed support, which locks the motor in place on the rig, and ensures that the motor body does not rotate. The previous back-support needed to be slightly strengthened and replaced, as the torque from the arm was too great for the original design.

Arm 2
The shoulder for the new arm needed to be designed from scratch. The new design was created having a different motor in mind, the brushless *Faulhaber 4490*. However it still used the previous design as a inspiration, and was therefore somewhat similar. The planned design, and the finished product can be seen in figure 5.4 below.


(a)

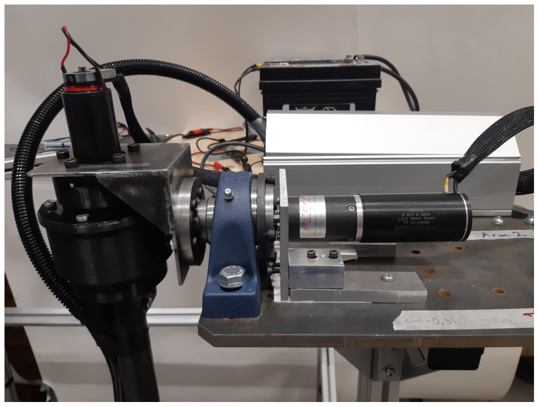

(b)

Figure 5.4: Side by side comparison between the CAD design and the real life implementation of the shoulder for arm 2.

A standard SKF *SY 507* house with *YAR 207-2F* bearing was placed upon the metal plate, and used as a starting point for the new design. Then a metal axis, meant to transfer the rotational movement and torque from the motor to the rest of the arm, was designed. This axis was manufactured using steel and was adapted to measurements on the bearings and the motor axis wedge.

Because the bearing was adjustable, some additional construction was needed to fixate it and to not put a dangerous strain on the motor axis. For this, the axis was adapted, an extra bearing was added and a steel ring was welded unto the bearing house.

For mounting the motor, a support was manufactured and adapted to the brushless motor, where the motor was fastened using four screws in the front. At first the support was printed in plastic, but it was later replaced by aluminum to increase the durability. The support was screwed unto the metal plate.

5.1.4 Elbows

The components on the old arm were replaced, and thus both arms had an identical design, based around the *Cheetah* motor. The finished design and implementation can be seen in Figure 5.5.

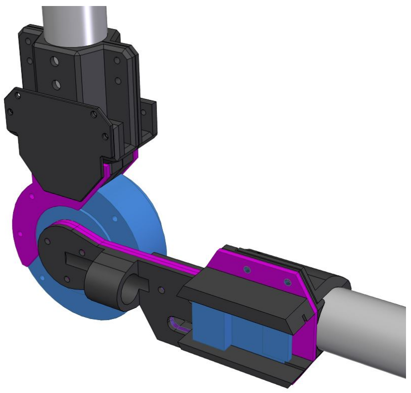
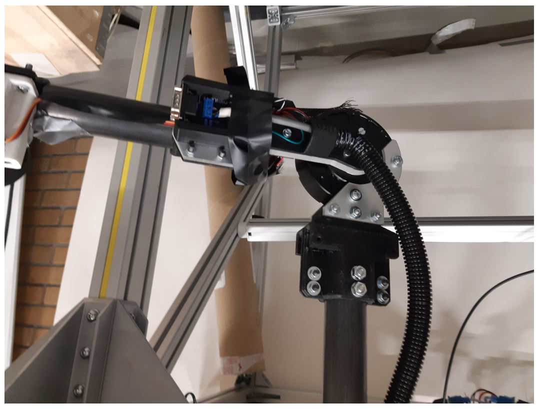



Figure 5.5: Side by side comparison between the CAD design and the real life implementation of the Elbow for both arms.

The design of the elbow was of critical importance to the success of the robot. The design needed to be light enough to be movable by the upper motors, yet still durable enough to withstand the torque and speed produced by the tossing motions. For this, a design of metal and plastic was produced, with carbon fiber tubing connecting the different parts.

Wherever force was expected to be applied, water-cut 3 mm aluminum pieces was used to provide a sturdy frame. Where no force was expected, light 3D printed material was used for connecting parts and attaching electronics. This can be seen in Figure 5.5 and Figure 5.6, where the black parts are plastic, and the purple are aluminum.

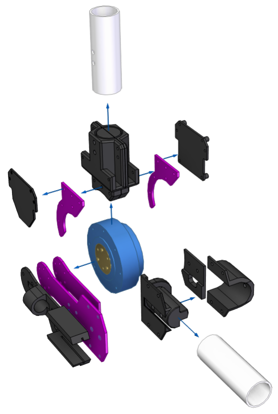


Figure 5.6: Exploded view of the elbow design.

5.1.5 End Effectors

The end effector provided the final aid to the throwing and catching process. The final design (seen in Figure 5.7) consisted of three main components: cone, lever and servo. The cone was a modified version of the one used in JuRP-2019. The thickness was increased, and a slit was added to allow the lever to move past the edge and lock the ball in place. The servo and lever combination added a gripping functionality to the end effector. All parts, excluding the servo, of the end effector were 3D printed in PLA plastic in order to keep the weight to a minimum.

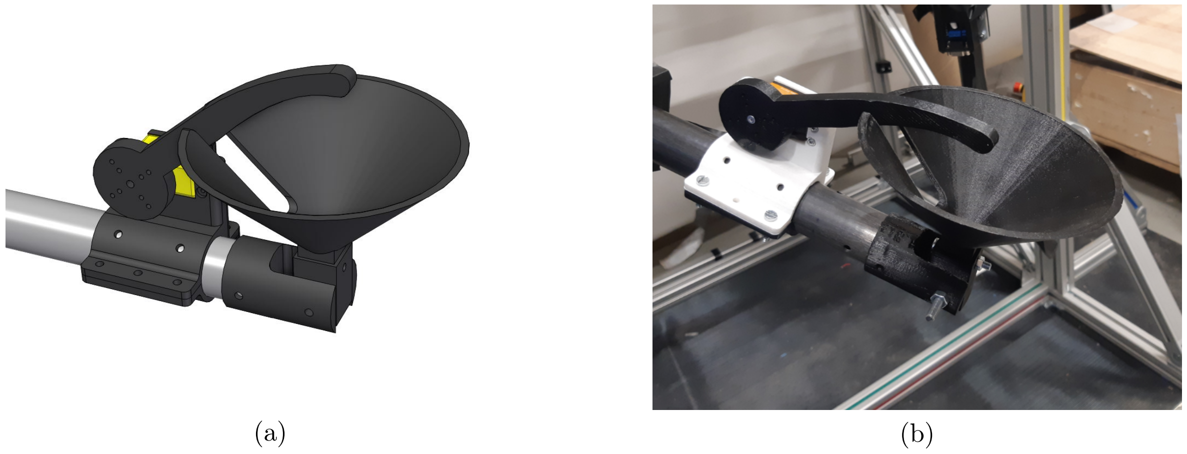

(a) (b)

Figure 5.7: Side by side comparison between the CAD design and the real life implementation of the end effector.

The new additional components weighed less than 200g, keeping the wight low. The mechanical complexity was also low as the only moving part, excluding the servo, was the lever. As a result, the movement was easily predicted and the potential risk of moving parts wearing out was kept low. The use of a servo made it easy to control the angle of the lever, as the servo angle was controlled by sending a PWM signals of different width to the servo via a micro controller.

5.2 Hardware and Electronics

5.2.1 Motors

Table 5.1 and Table 5.2 show the motors that were used in the final design for the respective arms, their corresponding drivers, and their placement.

Table 5.1: Listing of all the components used in arm 1.

Arm 1	**Motor**	**Gearbox**	**Driver**	**Max Torque**	**Max RPM**
Shoulder	Crouzet 8989B1-2	-	Pololu G2	12 Nm	74 RPM
Transverse	Faulhaber 3863CR	Faulhaber 38A	Pololu G2	-	-
Elbow	MIT mini Cheetah	-	-	17 Nm	382 RPM
Gripper	Savöx SC-1251	-	Arduino Nano	0.8 Nm	111 RPM

Table 5.2: Listing of all the components used in arm 2.

Arm 2	**Motor**	**Gearbox**	**Driver**	**Max Torque**	**Max RPM**
Shoulder	Faulhaber 4490 B	GPL042K-2V	Ingenia Pluto	9.59 Nm	154 RPM
Transverse	Faulhaber 3863CR	Faulhaber 38A	Ingenia Pluto	-	-
Elbow	MIT mini Cheetah	-	-	17 Nm	382 RPM
Gripper	Savöx SC-1251	-	Arduino Nano	0.8 Nm	111 RPM

Each motor chosen fulfilled the specifications and requirements for that joint detailed in chapter 4.

5.2.2 Hardware architecture

The system relied on drivers for the control of the motors. The previous year used a system mostly based on serial communication and PWM signals emitted from an Mbed Nucleo microcontroller, via Polulu drivers.

This year it was intended to implement a CAN protocol for controlling every motor, as this would increase the reliability and the speed of the system. This was successfully implemented on the second arm, and all of its motors were connected to the CAN bus. However, due to a lack of time, it was not entirely implemented on the first arm. Thus, the system relied on both a CAN protocol and serial to control the motors.

Figure[5.8] displays the hardware infrastructure and architecture implemented. The new architecture was fully installed on the new arm but with small deviations on the old arm, to facilitate the old existing solution.

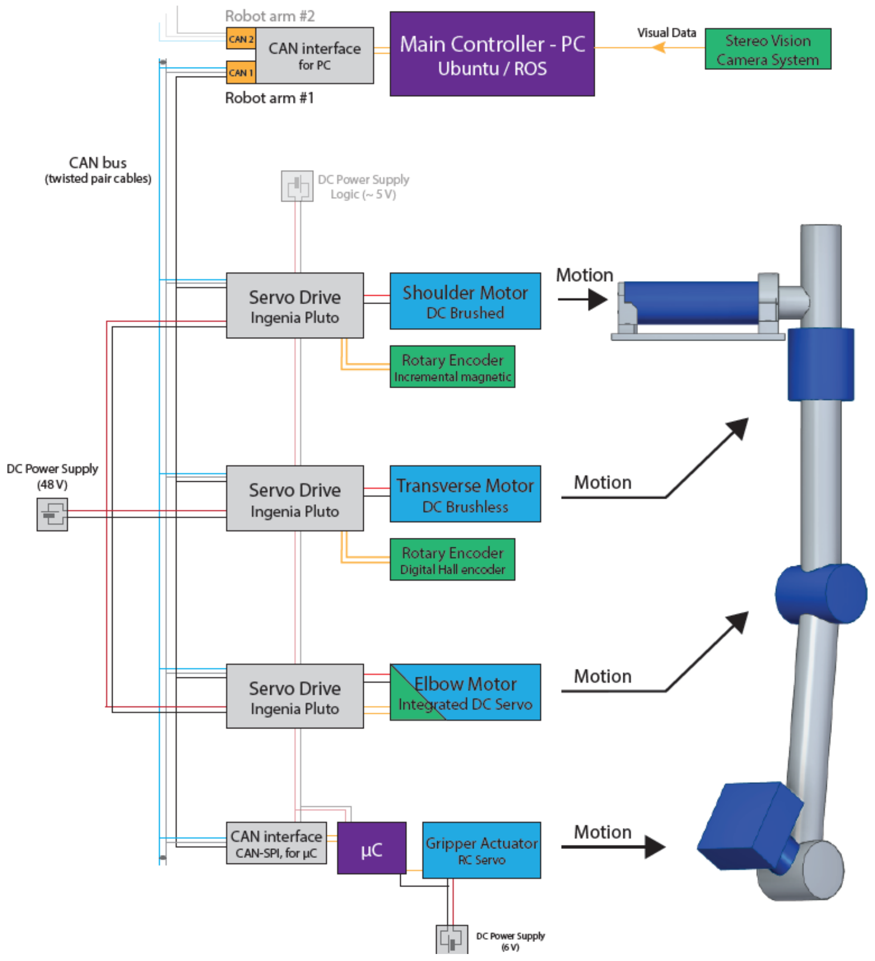

Robot arm #2
CAN 2
CAN 1
CAN interface for PC
Robot arm #1
Main Controller - PC Ubuntu / ROS
Visual Data
Stereo Vision Camera System
CAN bus (twisted pair cables)
DC Power Supply Logic (~ 5 V)
Servo Drive Ingenia Pluto
Shoulder Motor DC Brushed
Rotary Encoder Incremental magnetic
Motion
DC Power Supply (48 V)
Servo Drive Ingenia Pluto
Transverse Motor DC Brushless
Rotary Encoder Digital Hall encoder
Motion
Servo Drive Ingenia Pluto
Elbow Motor Integrated DC Servo
Motion
CAN interface CAN-SPI, for µC
µC
Gripper Actuator RC Servo
Motion
DC Power Supply (6 V)

Figure 5.8: The hardware architecture implemented to control the different motors

5.3 Software

5.3.1 Behavior Model

State machines was implemented to control the robot's different behaviors. The implemented scripts conducted either one or two arm juggling, throwing either across to the other arm, or catching the ball itself. User inputs shifted the different states. Once the throwing was accurate enough, the state shifting could be programmed to execute automatically after each other. The flowchart shown in Figure 5.9 illustrates the logic behind diagonally throwing one ball between the two arms. State machines were chosen over behaviour trees, due to its fast and simple implementation.

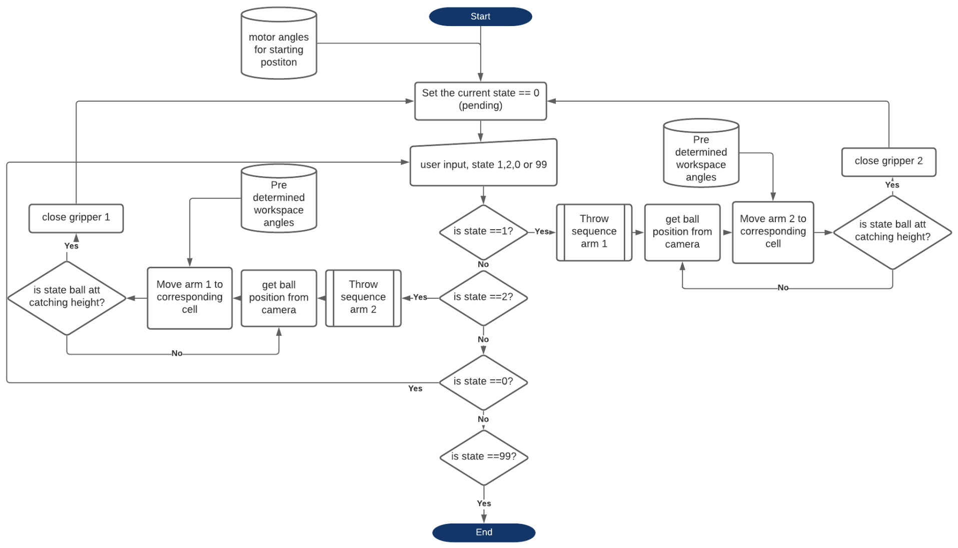

Start
motor angles for starting postiton
Set the current state == 0 (pending)
user input, state 1,2,0 or 99
Pre determined workspace angles
close gripper 2
close gripper 1
Yes
is state ==1?
Yes
Throw sequence arm 1
get ball position from camera
Move arm 2 to corresponding cell
is state ball att catching height?
No
Pre determined workspace angles
is state ball att catching height?
Move arm 1 to corresponding cell
get ball position from camera
Throw sequence arm 2
Yes
is state ==2?
No
No
is state ==0?
Yes
No
is state ==99?
Yes
End

Figure 5.9: A flowchart of the script handling juggling with one ball.

5.3.2 Throwing

Pre-programmed motor angles and pre-set waypoints executed the throwing sequence. The repeated motions created a high consistency in the landing point and enabled the workspace to be as small as possible. The angular acceleration from the elbow motor created the ball's vertical acceleration. The angular acceleration in the transverse motor gave the ball's necessary horizontal acceleration. The shoulder motor was fixated in 90(deg) during the throw sequence, decreasing the disturbance and uncertainty in the throw.

5.3.3 Catching

During simulations and later in the real implementation, the IKfast solver performed great with execution times between 5-20 ms for whole trajectories. However, the way the motion planner in Moveit was called, required a 300 ms long start up time for every new request. This led to an unacceptable delay in the end-to-end time. To avoid this delay, IKfast and Moveit were replaced with a pre-calculated inverse kinematics and a discretized planar grid for each arm's workspace. The calculated inverse kinematic was connected to each grid cell and stored in a python dictionary. Storing the predetermined inverse kinematics in a dictionary, gave the advantage of optimizing the kinematics operation to complexity O(1). The workspace was 30x35 cm and had a cell size of 5x5 cm. The script subscribed to the ball's position, and when the ball located itself over a cell, the script immediately sent the corresponding arm joint angles to the arm, see Figure 5.10.This solution opened up a lot of processing time from the ROS-packages that could be used for the vision system instead.

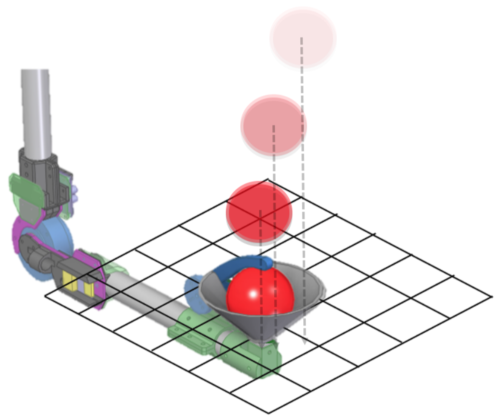


Figure 5.10: Illustration of how the balls position is connected to a certain cell in the workspace.

5.3.4 Robot Framework

The runtime peer-to-peer communication was handled by ROS (Robot Operating System), as it is on the verge of becoming an university-standard within robotics. Message-passing between processes and nodes was the main functionality for ROS in this project, as many of the low-level device controls and functionalities were too slow for a "real-time" system.

The main node that consisted of the state machine, subscribed on the ball's position (/ball_impact) topic, published by the camera node (/vision_node). The main node's state machine processed the ball's position and published the corresponding arm's motor angles for catching if it was in a "catching state". The motor angles were published on the topic "/motor_commands" which both the CAN bus-node (/pc2can_node) and USB node (/command_to_mbed) subscribed to. Every motor position was picked from the message and put on its matching bus (see Figure 5.11). There was also a homing topic that could send the machine to home position (/go_home).

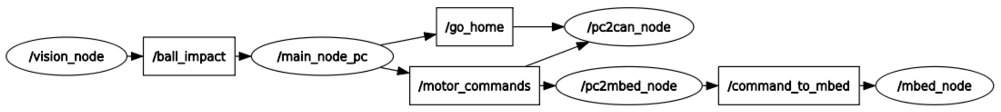


Figure 5.11: The ROS communication network, rectangles represents message topics and circles represents process nodes.

5.3.5 Motion Control

Servo drivers were integrated into all the new motors in the system. *Ingenia Pluto* servo drives controlled the new shoulder and transverse motor, using a PID controller, calibrated in their own software MotionLab. The open-source motors "MIT cheetah", used as the elbow motor, were controlled by an integrated servo driver. The firmware was adapted from the motor designer's (Ben Katz) GitHub and consisted of a current controller (PI-controller) with "Field Oriented Control" (FOC). The pre-existing shoulder and transverse-motor solution was kept, due to the project's time limitation. The old motors' P controller came from setting the I and D part in the PID library, downloaded from the microcontroller's webpage, to zero.

5.4 Vision System

The main goal of the vision system was to be able to identify the 3D-coordinates of the balls during juggling. Hence, the method- and design choices for each part of the vision system were made by considering its functionality, speed and accuracy. The different parts of vision include choice of camera, calibration, object detection, trajectory prediction and depth estimation. All were implemented in python, using OpenCV packages. The design alternatives and solutions for every part of the vision system will be described in this section of the report.

5.4.1 Calibration

The cameras used in this project came with compatible CS mount lenses. The cameras were placed approximately two meters in front of the rig. An appropriate lens was chosen after taking the following factors into consideration; the lenses should have a wide angle of view so that the camera could see the balls at all times during juggling, but it cannot be too wide because that will lead to an unnecessary intake of disturbing objects around the robot. Therefore, lenses with a 3.5 mm focal length was chosen in the final implementation process. These lenses were of the fisheye type, which produced visual distortion, i.e. it made straight lines appear curved. That is why the cameras needed to be calibrated.

Calibration was done by an OpenCV algorithm written in Python. It detects corner points and draws patterns as shown in Figure 5.12. The algorithm returns the camera's intrinsic and extrinsic matrices based on some free scaling parameter [24], which were used to transform the images, compensating for radial and tangential lens distortion [25].

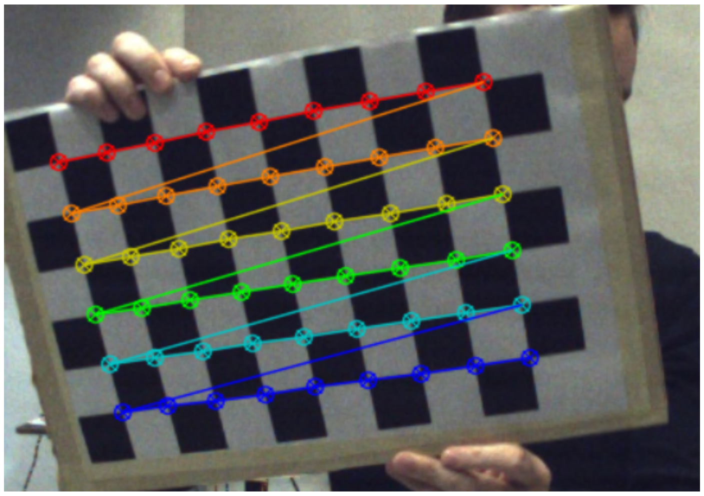


Figure 5.12: One image of chessboard with pattern drawn on it.

Approximately 100 images of a chessboard were taken and was then analysed by the algorithm to see whether corners were matched well enough as in Figure 5.12. The images were taken from different angles and distances to get as accurate intrinsic and extrinsic camera parameters as possible.

After calibration, a rectification which is the calibration between two cameras, needed to be done [32]. With the rectification of the stereo cameras, the vertical difference between the cameras decreased so that the left and right frames were positioned at the same vertical level. To make this possible, the stereo rectification found a rotation and translation vector, knowing each camera's corresponding point. Figure 5.13 shows the whole rectification process. From here, the images were now ready to be used in the tracking and depth calculating process.

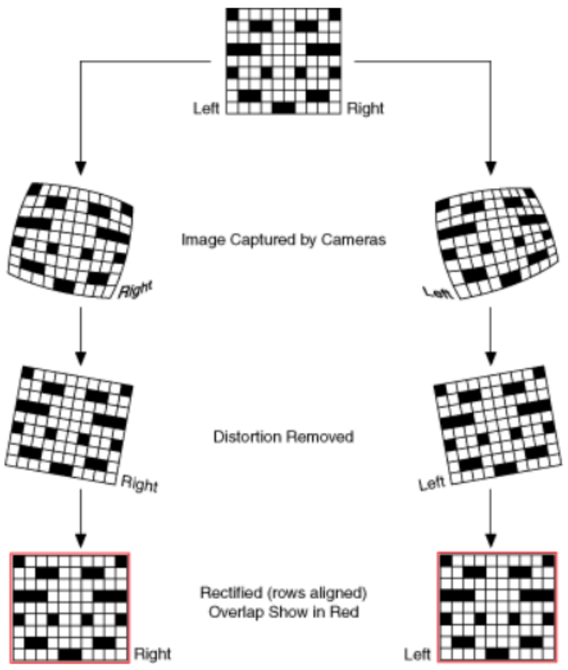


Figure 5.13: The process of rectification [32].

5.4.2 Sensor Choice

The motion capture chosen for this project ended up being of the markerless type. Two RGB cameras were chosen as suitable sensors for implementation of a stereo vision setup, allowing the system to have a perception of depth. Camera factors such as frame rate, interface throughput and focal length of the lens were considered when the camera choices were made.

There were two cameras that were considered as alternatives, both being RGB sensors. One camera was the *DFK 33UP1300*, borrowed from the sponsor company *Recab* and the other one was a *Dalsa Genie CR-GEN0-M1020* camera, provided by KTH. The Dalsa Genie only came with a lens that had a focal length of 8mm and the camera itself was capable of producing a frame rate at a maximum of 20 fps. Considering the fact that the DFK camera had a maximum frame rate at 210 fps and also had lenses with a focal length of 3.5mm, its performance was deemed as better than the Dalsa Genie one. Moreover, the Dalsa Genie camera used Ethernet as an interface which is slower than the DFK's USB3 interface in terms of throughput. Thus, due to these reasons, the DFK camera from Recab was picked as the final sensor choice.

5.4.3 Tracking Method

Choosing the right tracking algorithm is a vital part to be able to extract accurate 3D-positions. Especially the speed and precision plays a big role in the quality of the output. The methods that were considered were object detection with circle Hough transform and with HSV color isolation.

In order to choose a suitable tracking algorithm, it was especially important to take notice of the whole system environment, including the background wall, rather than the individual tracking objects. A part from the balls, the arms also had some circular dimensions in its motion. This made it hard to use methods such as background subtraction and circle Hough transformation alone. Color isolation with HSV however, was seen as a suitable option. Due to this choice of algorithm, other design alternatives such as color of arms and rig had to be chosen carefully. Balls in bright colors of red and green were chosen while the rest of the construction were chosen to be gray and black. A background sheet was also added behind the construction to allow for better tracking. After the HSV-filtering, the ball's 2D-coordinates in pixels could be extracted by finding the contour with the biggest area. This allowed for noises to be cancelled out but limited the amount of tracked balls with the same color to one. All of the above mentioned choices of methods were implemented in Python, using OpenCV-packages.

5.4.4 Depth Calculation

There were two alternatives when choosing the method to calculate depth. OpenCV had a function in the OpenCV-library where every point in the image was triangulated and thus depth was calculated for each point of the frames. A precise but relatively slow process. When testing, the depth was precise and accurate but the processing-time was too slow. The other alternative was a self-written script with only the ball in interest, so the triangulation calculation was only done for the ball center point instead of the whole frame. A fast but inaccurate process because of increased errors after the transformation of coordinates.

5.4.5 Coordinates Transform

The functions used for the transform were the following three:

$$Y = C \cdot \frac{Bf}{d} + k_1 \tag{5.1}$$

$$X = \frac{x_l Y}{f} + k_2 \tag{5.2}$$

$$Z = \frac{y_l Y}{f} + k_3, \tag{5.3}$$

where x_l and y_l is the left camera's coordinates in the horizontal and vertical axis, Y is the real-world depth from the cameras and f is the focal length of the lenses. C (=2.5) was a factor which handled the offset after the realization that the theory in which the equation was derived from did not match our real world result because of inaccuracy in the depth calculation method. k_1, k_2 and k_3 are constants which are given depending on where the origin is located. Note that the Y-axis in the camera coordinate system became Z, and vice versa, to match the coordinate system of the robot where Z points up and the Y-axis goes through the camera lens.

5.4.6 Trajectory Estimation

The choice of trajectory estimators was between an Euler-method-based estimator or a Kalman-filter-based estimator. The goal was to implement the fastest and most precise trajectory estimator. Neither of the alternatives were fast or precise enough to be implemented so the decision was made not to use an estimator at all and instead use a real-time solution, making the end effector always be positioned under the ball. This is far from ideal, but it was a better solution to have the arm lag behind the ball and sometimes catch it than have the arm go to a position derived from disturbance and that was a wrongly estimated then never be able to correct itself in time.

This was largely due to the inaccuracy of the depth calculation which made all the other coordinates inaccurate because of the dependency on depth when transforming the coordinates from the cameras to the real-world. From the testing performed, Kalman-filter seemed like the preferred way going forward as the accuracy was superior.

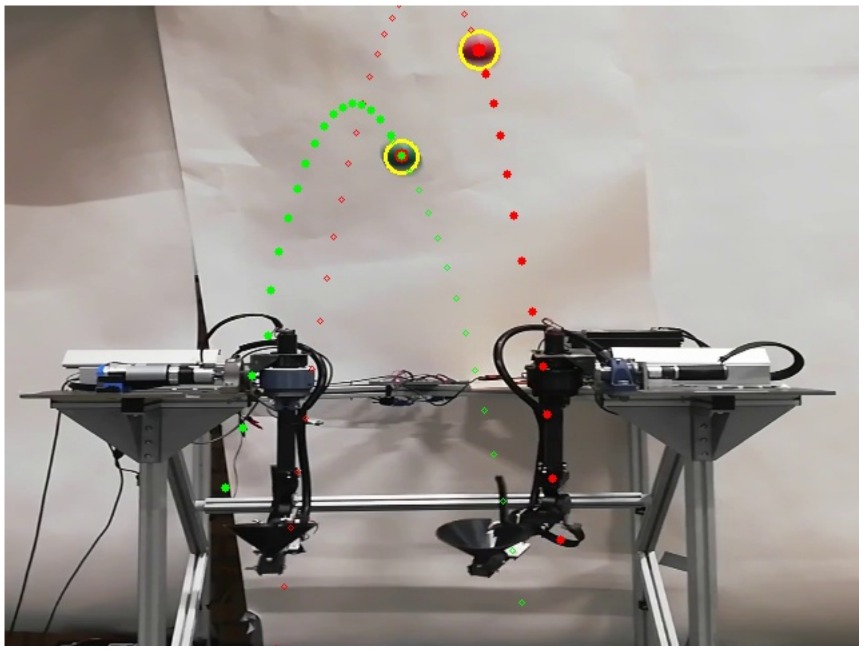


Figure 5.14: The Trajectory estimation of two balls in different colors, using a kalman filter

5.5 Communication

Communication in decentralised control system (controllers integrated with servo drivers etc.) requires robust communication protocols for sending all different commands and handling signals. Two different types of buses were implemented on the system: USB, and CAN. There were one USB bus from each of the two cameras to the computer and also one bus between the microcontroller and the computer. The remaining devices were connected to the CAN bus.

5.5.1 USB

USB was chosen for the Vision-system cameras and microcontrollers for practical purposes since it was the only already hardware-supported protocol. The cameras were transferring frames at approximately 210 fps on the USB bus to the host computer. A ROS node handled the unpacking using the C API for video I/O together with the Open Source Computer Vision library OpenCV.
The USB bus between the computer and microcontroller served as a link in point-to-point ROS serial transmission line communications. The protocol adds a packet header and tail, which allows multiple topics to share a common serial link [27]. The bus was transmitting and receiving angle positions for the shoulder and transverse motors on arm 1, packaged in a ROS geometry_msgs/Vector3.msg format.

5.5.2 CAN

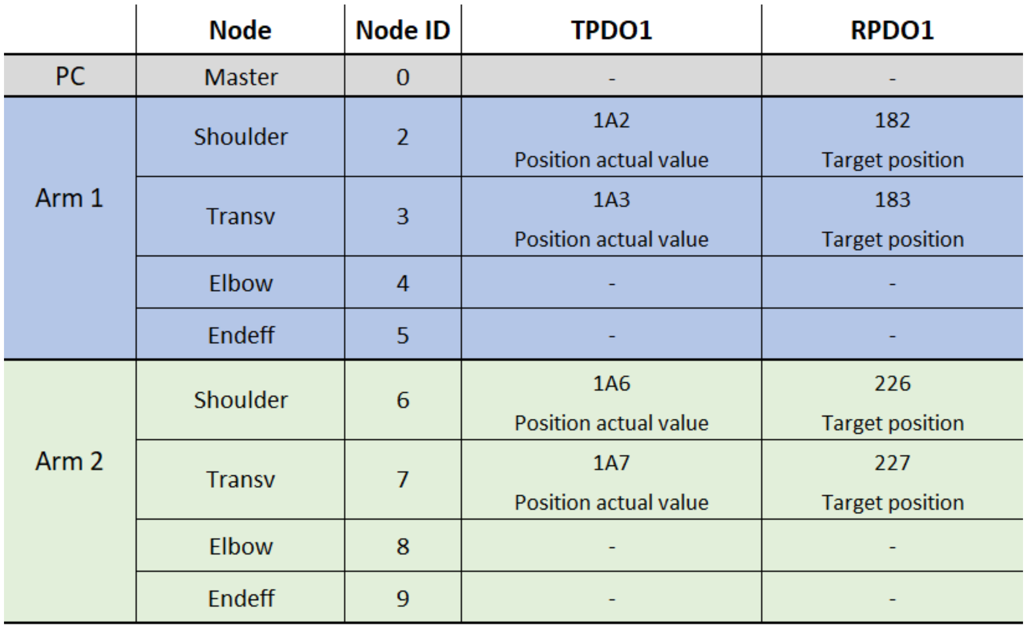
For the servo drivers, a CAN bus was implemented, due to its: robustness, daisy-chaining capability, and good cable management between the PC and robotic arms. A USB-to-CAN interface (see Figure[5.15]) was used to connect the main computer to the CAN bus. The standard CAN bus (CAN 2.0 B) was used, with an 11-bit identifier along with an 8-bit data frame. The bus was configured for 1 Mbit/s data rate, and all nodes in the network connected in a daisy-chain.

Figure 5.15: USB-to-CAN interface [15].

Each of the actuators (nodes) received different commands depending on the type of controller connected.
The *Ingenia Pluto* servo drivers supported the standard CANopen protocol together with CiA 402 [7] for motion control. The Process Data Object (PDO) was configured to be able to read and write position-data from the node controllers with high frequency.

The end effectors' CAN-transceiver and the *"MIT cheetah"* elbow actuators were implemented with the simple CAN configuration, without the standard CANopen protocols and dictionaries, but using the same signal framework. In practice, the end effectors only needed to receive a boolean 1/0-state for controlling the gripper and was only programmed to store their individual Node-ID for network filtering purpose, and then read just one bit of data in the messages passing the filter. The elbow-actuators also used the standard CAN overhead but packed the data in a more compact format, which can be read in more detail in [17], page 48.

The PC node acted as a Master or Network manager controlling each of the nodes operational states and was responsible for sending/receive and transmit-commands to the network nodes. The node identification was of great importance as it gave hierarchy to the messages (see configuration of the final implemented network in Figure[5.16].



	Node	Node ID	TPDO1	RPDO1
PC	Master	0	-	-
Arm 1	Shoulder	2	1A2 Position actual value	182 Target position
	Transv	3	1A3 Position actual value	183 Target position
	Elbow	4	-	-
	Endeff	5	-	-
Arm 2	Shoulder	6	1A6 Position actual value	226 Target position
	Transv	7	1A7 Position actual value	227 Target position
	Elbow	8	-	-
	Endeff	9	-	-

Figure 5.16: A table of the CAN configuration

Chapter 6

Ethics and Environment

6.1 Ethics

The group members of the project were upholding the code of conduct provided by KTH, promoting:

- Consider all people to be of equal value.
- A respectful and inclusive work and study environment free from harassment, sexual harassment, discrimination and offensive behaviour.
- Leading by example and maintain a professional approach to all meetings and in all my communication.

6.2 Environment

The project was trying to work according to the UN goals for sustainability. Goal 12 (Responsible consumption and production) was the UN sustainability goal that concerned the project the most. The project tried to reduce waste by choosing durable materials that wouldn't break. Also, collecting old components from retired projects at the department helped to reduce the need for consumption.

Chapter 7

Verification and Validation

This year, the two main goals were to design and build 1) *a mobile prototype*, that is 2) *capable of juggling in a human-like manner*. Both goals were continuously in mind while designing the robot. Verifying what "human-like" juggling meant was perceived as a subjective goal and the same went for "mobile". The next set of goals could be evaluated using hard values. 3) *Design and construct a functional prototype*, 4) *for juggling two-three balls*, 5) *using some device to grip and release the balls, making it autonomous.* Goal 3), 4) and 5) could simply be answered by the performance of the final prototype. The goals could also be divided into smaller sub-goals that were more measurable. One of which was the accuracy of the throws.

A good feedback loop and fast end-to-end time were important to consider if the robot were to juggle successfully. The throws were never going to be consistent enough that the catching sequence could rely solely on a feedforward loop and that was also not the purpose of the project.
One way to verify the juggling was to measure the end-to-end time in the feedback loop. The actual end-to-end time depended on several factors, such as different timings in low-level control loops and when the ball was in the field of view of both cameras. A worst-case end-to-end time could be calculated by adding the maximum time from the subsystems and the communication time between them. From a ball being thrown into the view of the cameras to the motors being actuated the following steps need to be executed:
Read the camera frames, identify the ball, convert 2D pixel coordinates to 3D coordinates, send those coordinates to the high-level control loop, map the coordinates to motor position via the pre-calculated inverse kinematics solver, send the motor positions to the microcontroller, the microcontroller needed to read the current position, calculate the appropriate PWM-signal and send that to the motor drivers.

The execution time of the whole chain of events were crucial to have a fast and good performance.

Chapter 8

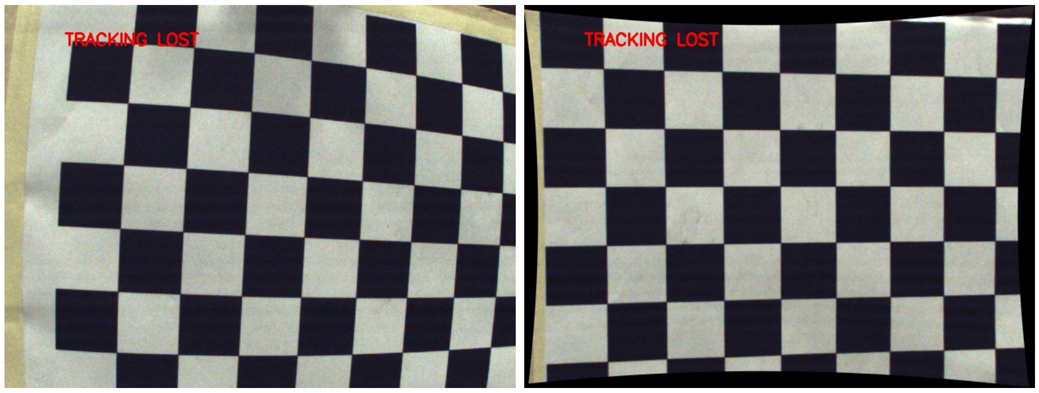
Results

8.1 Juggling

The final prototype was at its best able two juggle one ball up to seven times in a row, and was also able to throw and catch two balls at the same time once without dropping any of the balls. However, the software was not stable enough towards disturbance, so occasionally the precision and repeatability went down, and balls were missed.

8.2 Vision System

Calibraiton
Out of the 100 pictures that were taken for each camera, about 70 of them were approved by the algorithm and used to find the intrinsic and extrinsic parameters. The results from the calibration can be seen in Figure 8.1 which represents before (a) and after (b) calibration was applied.



(a) Before calibration (b) After calibration

Figure 8.1: Calibration of visual distortion.

Filter Results

The results of using color isolation as an object extraction method can be seen in Figure 8.2. The two upper pictures is a representation of what the frames look like before masking them with HSV-filter, while the lower pictures show the same frames with added filter.

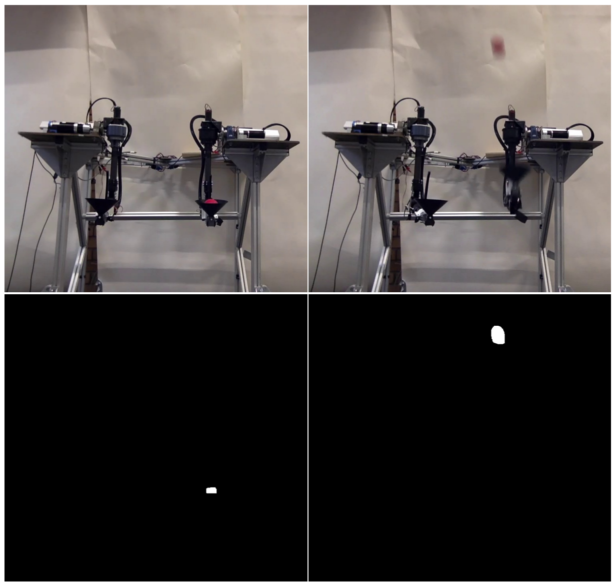


Figure 8.2: A representation of frames before and after the filtering process. The two upper pictures correspond to the two lower ones.

Trajectory Estimation

Figure 8.3 below shows a visualization of Kalman-filter estimation to the left and Euler forward estimation to the right. The pictures were mostly for visualization when testing and comparing.
The camera system succeeded in providing an accurate function-output for the ball's point in the 3D space, shown to be the most time-efficient way of tracking the ball in a fast and precise manner. Testing showed the Kalman-filter to be the fastest alternative that repetitively produced correct trajectories and was the more accurate method between the two.

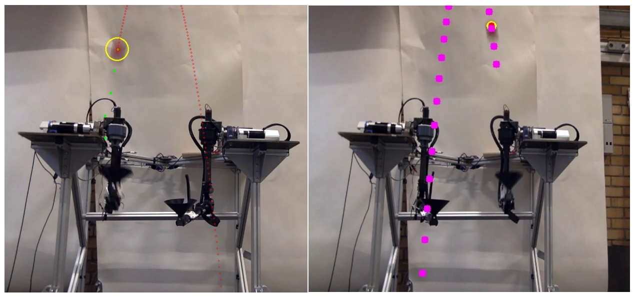


Figure 8.3: Visualizations of the trajectory estimation methods Euler estimation (left) Kalman-filter estimation (right).

8.3 Control System

To achieve reliable juggling with feedback from a vision system the end-to-end time was an important factor. Table 8.1 shows the execution and end-to-end time for Arm 1 (old arm using USB) with the mBed microcontroller.

Table 8.1: Worst-case execution and end-to-end times.

Subsystem	Time [ms]	Total share [%]
Vision system	41.6	53.9
High level control	10.2	13.2
Inverse Kinematics	0.9	1.2
Rosserial (USB communication)	14.0	18.1
Low level control loop	10.0	13.0
Read sensor value	0.005 x 2	0.01
Actuate Motors	0.22 x 2	0.06
Total End-to-End	77.2	

In the vision system, everything from reading the camera frames and calculating the 3D coordinate added to the execution time. The High-level control consisted of the internal communication in ROS, selecting state in the state machine, and sending out the values from the inverse kinematics solver. The inverse kinematics node included the equations to calculate the corresponding workspace index in the pre-set grid to the desired catching position and retrieving motor positions from the look-up table. The "ROS_serial" package generated a lot of the time for the communication between the PC and the microcontroller. The low-level controller ran with 100 Hz which gave a worst-case execution time of 10 ms. The final part consisted of reading the two encoders and actuating the two motors. In the proposed software, the Vision system made up more than half the end-to-end execution time and the rest of the process was divided between the High-level control, Low-level control loop, and the communication between those two.

Chapter 9

Discussion and Conclusions

Kinematics and Motion
The usage of ROS can be debated. The performance of the ROS plugins and packages for visualization, motion control, and kinematics didn't perform as fast as required. In the end, ROS was only used for handling the communication between nodes, which easily could have been handled in a Python or C++ script. The standard ROS-packages are built to be as general and applicable as possible, which slows down the system. A better solution for motion planning would be to write a more specific code for the JuRP machine, consisting of a simpler motion planner. For example, a Discredited 3D workspace with reasonable angle resolution of each arm and implement a "A-start" planning algorithm would probably be sufficient enough. The better the precision of the throw is, the smaller the workspace can be (or higher resolution for smoother motion). Since each arm only uses three motors for its position, the inverse kinematic can be calculated for each robot pose in high frequency with an analytic algorithm. However, for the ROS packages to be sufficient enough, it would be recommended to try the new beta version of ROS 2 that supposedly supports real-time systems. Note that the implemented pre-calculated inverse kinematics solver worked very well for 1 ball and 2 ball-juggling, for 3 ball-juggling the inverse kinematics was not tested. The suggested solution of inverse kinematics solver may run into problems if the workspace is to be expanded so that the whole motion, including both throwing and catching, can be planned in real-time.

Modeling and Simulation
Very little time was put into modeling and simulating the forces and motions of the robot. Since the previous groups had put a lot of effort into calculations, this year's group felt it was unnecessary to redo all the work and instead use the calculations and simulations as a basis for material and electronic procurement. This year's group was aware of the risk of miscalculations. Hence, All the motors, load-bearing framework, force strained links, and current leading electronics were over-dimensioned to compensate for the uncertainty in calculations. This was extremely important in regards to manufacturing the robot arms. The old arm design didn't withstand the accelerations of a throw, all the load-bearing 3D-printed parts broke and the form-fitted joints lost their positions, leading to malfunctions and dangerous loss of positioning. It became very clear during the development stage that a robust hardware and electronics design was key for this project.
The main use of simulations was to try different motion planners and inverse kinematics solvers and to see the response of the robot arms when trying to catch a ball or move to a desired position. However, a whole juggling sequence was not simulated due to difficulties with making the simulations realistic enough with free moving juggling balls and implementing a camera system. Creating a near realistic simulated world would be of great use. It would allow all parts of the software to be tested and developed without having to worry about hardware failures, both when it comes to troubleshooting and to not expose the robot to unnecessary wear and fatigue of mechanical and electrical parts.

Vision System

From a speed and accuracy point of view, there are some issues present. The depth calculations is the function that takes the longest time to execute, as it makes a depth map for all pixels on the ball. As a result, calculating the depth using point values of the ball's center point is chosen as the final method which leads to a faster system but at the expense of less accuracy.
There is also a need for adding compensating factors to the depth value, to convert it to real-world coordinates when using the current method. Because of this change, the trajectory gets affected as well. The precision of the trajectory estimator decreases with the camera distance for the current method as well. Furthermore, there is a slight but noticeable difference in the camera quality. One camera appears to suffer from internal noise which slightly affects the output of the ball detection. Because of this camera flaw, at times the tracker output is not constant when the ball is still. However, this error in the position values of the object was insignificantly small and doesn't visibly affect the robot's performance.

The threshold values in the HSV-filter have to be updated at times due to irregular light-exposure in the room. The amount of light present in the room makes noticeable changes to the shades of color on the ball registered by the cameras. Despite these issues, the errors can be worked around. By including additional constants to the depth output-values as earlier mentioned and by updating the HSV-filter when necessary. The vision system can sufficiently be used in the final product and contributes to the achieving goal of automatic juggling.

Wiring
Changing the wiring for the end effector is key. For the moment being, all the most sensitive wiring is positioned in the absolute least preferable place. A lot of jerky and high acceleration motions is performed in the elbow joint during the throwing sequence. This wiring placement is the main source of malfunctioning during the machine's run time. Cables are being either destroyed or ripped out of their sockets. An improvement would be to move the end effector's driver (Arduino Nano) and P-mod to the shoulder platform and just have three wires going down to the servo motor. The slightly longer distance in between the driver and servo would give a less stable drive, but since the performance of the end effector has the highest tolerance, with little to no required precision, the benefits would truly outperform the disadvantages. Also, the drive of the end effector was successfully proven to work flawlessly with long cables and a displacement of approximately two meters during the development stage.

Juggling
Juggling is, as expected, a very challenging task for a robot to achieve. With this in mind, the performance of the resulting solutions for both hardware and software worked very well for juggling with one ball and acceptably well for juggling with two balls. The robot's record for juggling with one ball is seven continuous throws and catches. That points towards a desirable consistency for all subsystems. Also, reliability and robustness in the integration and collaboration between the subsystems.
Juggling with three balls was barely tested in this project, but three balls were simultaneously thrown into the air to confirm that the robot arm's speed and the low-level controllers are capable of juggling three balls. This test showed successful results in terms of mechanical speed. However, the end effector, high-level control, and vision system were not built to handle this situation. It is believed that with small changes to the parts mentioned above, the current robot could achieve 3-ball juggling.

Chapter 10

Future Work

As this project, like most, had a limited time frame there are still a lot of aspects to investigate an improve. This chapter will present some thoughts on possible future improvements or investigations that could be interesting and/or beneficial.

10.1 Structural Changes

Construction Design
One major change that could be added to the system would be to increase the numbers of degrees of freedom. This would be done by adding an additional rotation in the upper parts of the arm, allowing for more human-like movement of the entire arm. This could enable more possibilities in reaching different coordinates. However, this would become a much more complicated system when it comes to inverse kinematics.

End Effector
To be able to reliably handle and juggle more than two balls, a rework or modification on the end effector would be needed. Possibly another lever could be added and/or modification on the cone could be made in order for the end effector to be able to hold more than one ball at a time.

For the robot to increase the space, in which catching the ball is possible, another degree of freedom near the end effector would be beneficial. This would be useful for positions where the end effector would have otherwise dropped the ball or be unable to catch it due to the angle of the cone. Adding this functionality would likely mean adding components, and thus adding weight. This could require a complete re-evaluation of earlier calculations of motor requirements as well as strength of the arm, so great consideration is needed. JuRP-2019 made a solution with two extra degrees of freedom for the end effector. However in the end it was discarded, as the weight of the end effector became too heavy for the motors in the arms.

Software
An alternative to running the software/system on a regular PC could be to run it on the already provided Nvidia Jetson X1, which is specifically made for the latest visual computing applications. This year's project group had problems installing the correct drivers and also had integration problems with the cameras, which is the reason why it was not used.

To solve many problems regarding time, a solution could be to make the system run on a real-time operating system with real-time robotics middleware. ROS2 was recently released with real-time support which could be a good update to the system as well as using a real-time operating system. Lack of guides and tutorials and enough problems with ROS1 was the reason for not upgrading to real-time within this project round.

10.2 Incremental Changes on Current System

Construction Design
Replacing the bearing house and motor mount on shoulder arm 2 could be necessary. It is welded in cast iron, and will most likely suffer failure in time. While the angle iron and transverse parts are working fine, the axis and the bearing houses could have better tolerances, which could make for a less jerky system. The aluminum support which the motor is attached to is also sub-par work, and should be replaced with something that has been more carefully constructed.

Due to the necessary distance increase, the camera mount on the rig is currently fastened to the floor with the help of tape. A more permanent extension with the help of additional aluminum profiles and associated accessories could be beneficial.

Hardware
It would be beneficial to streamline the shoulder motors as to increase symmetry. Most likely the best choice would be to update the current brushed DC motor on the old arm to the same brushless DC motor used in the second arm, or to a similar one.

As of right now, the shoulder and transverse motors on the old arm are driven by a microcontroller with the help of two Polulu drivers. For the sake of much faster e-2-e-communication and symmetry, this solution should be changed to Ingenia servo drivers as well and put on the CAN bus. This should be a simple task since the wiring is already done and the drivers are already procured and prepared. The Ingenia calibration is done in a free to download software called "MotionLab". Don't forget to detach the arm and/or sensible parts before configuring the motors with the new Ingenia drivers. The wrong sensor settings may lead to that the motors will spin uncontrollably during the torque calibration and could damage the robot arm if not handled carefully (this was learned the hard way).

The Cheetah motor, located in the elbow, fulfilled all its requirements. However, due to the open-source firmware and internal driver, it can be problematic to control, especially since there doesn't exist any documentation. Replacing it with a similar motor (T-motor AK80-6) with more easily accessible control and documentation could be considered.

Software
A solution to achieve a more structured and model-based coding structure would be to change the current State Machine to a Behaviour Tree. It would enable multiple group members to work on subtrees simultaneously, and also making it easier to add and subtract functions as the project goes along.

The current hardware solution for translating CAN signals to SPI signals for the end effector is too complicated. It could be simplified to one PCB instead of P-mod to Arduino, and Arduino to servo.

The inverse kinematics is for the moment being not plugged in. The moveit plug in "IKfast" was performing fast enough with the worst case planning time at approximately 0.002 s. However, the ROS moveit package that has to be implemented for the planning was not fast enough. The package's fastest planner "RRT" had an average performance time of 0.3 s and wasn't suitable for real time planning. A Solution for this problem could be to implement a self made planner and inverse kinematics solver that is more specific for this project. An IK-solver was written for this project but chosen not to be used since the hard coded positioning solution was deemed to be the best option with the means available, since it had the fastest performance.

About the juggling, any kind of learning using AI, like reinforcement learning, that can give a good prediction of where the balls will land would be great.

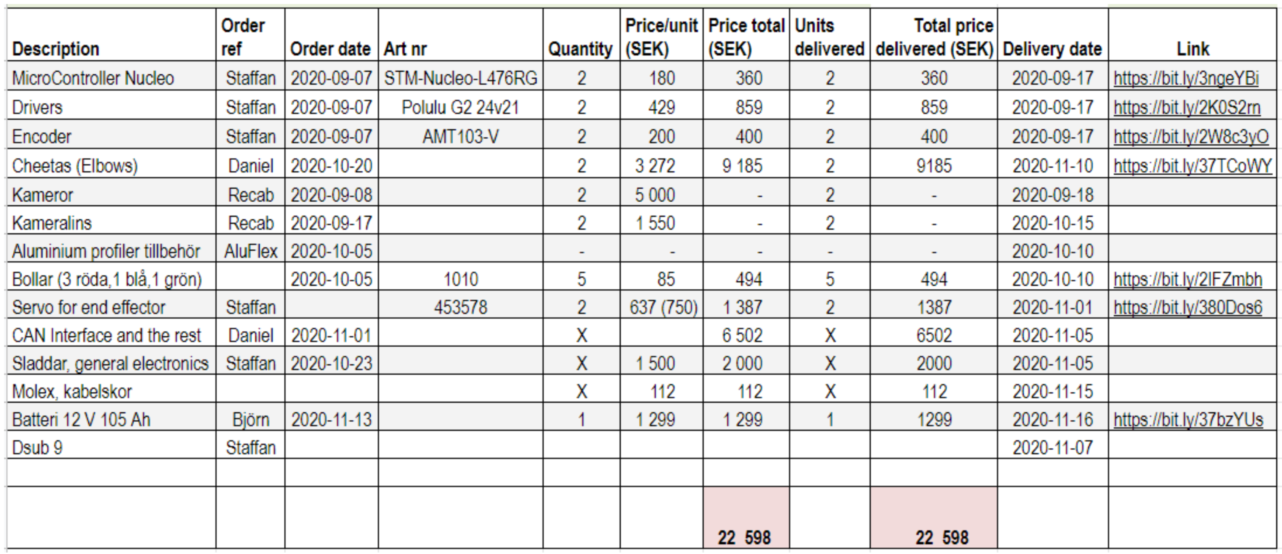
Vision System
In short, there was a speed versus accuracy compromise for the different parts of the vision system. High precision can be achieved without decreasing the speed of the system to a problematic level, by for example writing the code in C instead of Python. The code script can also be executed on a GPU instead of CPU to further achieve greater speed. This can allow the computationally heavy depth map to be used and thus achieve a higher precision and speed in the overall vision system. Furthermore, the camera that suffered from noise can be replaced in order to avoid vibrations in the object tracker, the result being a more stable and constant output position than what was achieved in this project. An implementation of the trajectory estimator can also be experimented with. Because of the slow execution time of the vision system and the low accuracy on the depth estimation, the trajectory predictor was excluded in the final presentation. However, with enough computational power, the trajectory estimator may be implemented with the rest of the system as well.

Chapter A

Appendix

A.1 Appendix A

JuRP20 Budget



Description	Order ref	Order date	Art nr	Quantity	Price/unit (SEK)	Price total (SEK)	Units delivered	Total price delivered (SEK)	Delivery date	Link
MicroController Nucleo	Staffan	2020-09-07	STM-Nucleo-L476RG	2	180	360	2	360	2020-09-17	https://bit.ly/3ngeYBi
Drivers	Staffan	2020-09-07	Polulu G2 24v21	2	429	859	2	859	2020-09-17	https://bit.ly/2K0S2rn
Encoder	Staffan	2020-09-07	AMT103-V	2	200	400	2	400	2020-09-17	https://bit.ly/2W8c3yO
Cheetas (Elbows)	Daniel	2020-10-20		2	3 272	9 185	2	9185	2020-11-10	https://bit.ly/37TCoWY
Kameror	Recab	2020-09-08		2	5 000	-	2	-	2020-09-18	
Kameralins	Recab	2020-09-17		2	1 550	-	2	-	2020-10-15	
Aluminium profiler tillbehör	AluFlex	2020-10-05		-	-	-	-	-	2020-10-10	
Bollar (3 röda,1 blå,1 grön)		2020-10-05	1010	5	85	494	5	494	2020-10-10	https://bit.ly/2IFZmbh
Servo for end effector	Staffan		453578	2	637 (750)	1 387	2	1387	2020-11-01	https://bit.ly/380Dos6
CAN Interface and the rest	Daniel	2020-11-01		X		6 502	X	6502	2020-11-05	
Sladdar, general electronics	Staffan	2020-10-23		X	1 500	2 000	X	2000	2020-11-05	
Molex, kabelskor				X	112	112	X	112	2020-11-15	
Batteri 12 V 105 Ah	Björn	2020-11-13		1	1 299	1 299	1	1299	2020-11-16	https://bit.ly/37bzYUs
Dsub 9	Staffan								2020-11-07	
						22 598		**22 598**		

A.2 Appendix B

Full List of Components used in final product

Motors

- Crouzet 8989B1-2
- Faulhaber 3863 CR
- FAULHABER 4490
- MIT Cheetah
- Savöx Standard SC1251MG Digital servo

Drivers

- Polulu
- Ingenia Pluto C

Encoders

- AMT 102-v
- AMT 103-v

Micro-controllers

- Mbed Nucleo L476RG
- Arduino Nano

Computer

- DELL ...

Cameras

- DFK 33UP1300 with 3.5 mm lens

CAN Accessories

- Ixxat USB-CAN V2 INTERFACE 1.01.0281.12001

Table A.1: DH parameters.

Frame	θ	α	r_i	d
1	θ_1	-90	0	a_1
2	θ_2	90	0	a_2
3	θ_3	0	a_3	0

A.3 Appendix C

Forward Kinematics and Jacobian

The Frame transformations in accordance to the DH-process.

$$H_{01} = \begin{bmatrix} Cos(\theta_1) & -Sin(\theta_1)Cos(-90) & Sin(\theta_1)Sin(-90) & 0 \\ Sin(\theta_1) & Cos(\theta_1)Cos(-90) & -Cos(\theta_1)Sin(-90) & 0 \\ 0 & Sin(-90) & Cos(-90) & a_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

$$H_{12} = \begin{bmatrix} Cos(\theta_2) & -Sin(\theta_2)Cos(90) & Sin(\theta_2)Sin(90) & 0 \\ Sin(\theta_2) & Cos(\theta_2)Cos(90) & -Cos(\theta_2)Sin(90) & 0 \\ 0 & Sin(90) & Cos(90) & a_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

$$H_{23} = \begin{bmatrix} Cos(\theta_3) & -Sin(\theta_3)Cos(0) & Sin(\theta_3)Sin(0) & a_3Cos(\theta_3) \\ Sin(\theta_3) & Cos(\theta_3)Cos(0) & -Cos(\theta_3)Sin(0) & a_3Sin(\theta_3) \\ 0 & Sin(0) & Cos(0) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

The transformation matrix T between the base frame and end-effector, calculated in Equation A.4 by matrix multiplication between the transformation matrices H_{01}, H_{01}, and H_{01}.

$$T = H_{01} \cdot H_{12} \cdot H_{23} \tag{A.4}$$

The Jacobian could be calculated with either the DH-Algorithm A.2 or using built in functions in python.

Table A.2: Algorithm for Jacobian calculations.

Linear or Rotational	**Prismatic**	**Revolute**
$[\dot{x}\ \dot{y}\ \dot{z}]$'	$\mathrm{R}^0_{i-1}[0\ 0\ 1]'$	$\mathrm{R}^0_{i-1}[0\ 0\ 1]' \cdot (d^0_n - d^0_{i-1})$
$[\omega_x\ \omega_y\ \omega_z]'$	$[0\ \ 0\ \ 0]$'	$\mathrm{R}^0_{i-1}[0\ 0\ 1]'$

Bibliography

[1] *ABB IRB120.* URL: https://new.abb.com/products/robotics/industrial-robots/irb-120/irb-120-cad (visited on 05/12/2020).

[2] E. W. Aboaf, S. M. Drucker, and C. G. Atkeson. "Task-level robot learning: juggling a tennis ball more accurately". In: *Proceedings, 1989 International Conference on Robotics and Automation.* 1989, 1290–1295 vol.3.

[3] J. Arrizabalaga et. al. *The Juggling Robot Project.* JuRP2019. 2019.

[4] C. G. Atkeson et al. "Using humanoid robots to study human behavior". In: *IEEE Intelligent Systems and their Applications* 15.4 (2000), pp. 46–56.

[5] "Baseline and Triangulation Geometry in a Standard Plenoptic Camera". In: Jan. 2018, pp. 21–35. DOI: 10.1007/s11263-017-1036-4.

[6] C.Harris and M. Stephens. "A COMBINED CORNER AND EDGE DETECTOR". In: (1988).

[7] CAN in Automation CiA. *CANopen device profile for drives and motion control.* 2020. URL: https://www.can-cia.org/can-knowledge/canopen/cia402/.

[8] *Depth calculation and Coordinates Transform.* URL: http://www.cad.zju.edu.cn/home/gfzhang/training/stereo/04stereo.pdf.

[9] *Euler method.* URL: https://en.wikipedia.org/wiki/Euler_method.

[10] Katie Gatto. *A robot that can juggle five balls (w/ video).* 2011. URL: https://phys.org/news/2011-06-robot-balls-video.html (visited on 04/22/2020).

[11] David D. LukeDavid C. Gilbert. *Universal serial bus peripheral bridge with sequencer.* 2003. URL: https://patentimages.storage.googleapis.com/78/4a/2f/2d3996aef589d1/US6505267.pdf (visited on 12/11/2020).

[12] Rob Goodman and Jimmy Soni. *CLAUDE SHANNON: MATHEMATICIAN, ENGINEER, GENIUS... AND JUGGLER?* 2017. URL: https://www.juggle.org/claude-shannon-mathematician-engineer-genius-juggler/ (visited on 04/22/2020).

[13] *How does Light Influence 3D Scanning?* 2020. URL: https://www.revopoint3d.com/how-does-light-influence-3d-scanning/ (visited on 05/16/2020).

[14] I. Ishii et al. "2000 fps real-time vision system with high-frame-rate video recording". In: *2010 IEEE International Conference on Robotics and Automation.* 2010, pp. 1536–1541.

[15] *Ixxat.* URL: https://www.ixxat.com/products/products-industrial/can-interfaces/usb-can-interfaces/usb-to-can-v2-professional?ordercode=1.01.0281.12001 (visited on 12/11/2020).

[16] *Kalman filter.* URL: https://en.wikipedia.org/wiki/Kalman_filter.

[17] B. Katz. "A Low Cost Modular Actuator for Dynamic Robots". In: *Massachusetts Institute of Technology* (2016).

[18] Young-Bum Kim et al. “Multi-Player Virtual Ping-Pong Game”. In: Dec. 2007, pp. 269–273. ISBN: 0-7695-3056-7. DOI: 10.1109/ICAT.2007.34.

[19] J. Kober, M. Glisson, and M. Mistry. “Playing catch and juggling with a humanoid robot”. In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. 2012, pp. 875–881.

[20] Kungliga Tekniska Hogskolan KTH. *CAN bus.* Creative Commons Attribution-Share Alike 3.0. URL: https://www.kth.se/social/upload/526eab8ef2765479ddbd9131/CAN.

[21] Ioanna Mitsioni. *Introduction to Robotics DD2410, 2020,KTH ROYAL INSTITUTEOF TECHNOLOGYn.* 2020. URL: https://www.kth.se/student/kurser/kurs/DD2410?l=en (visited on 12/11/2020).

[22] L. Mündermann, S. Corazza, and T.P. Andriacchi. “The evolution of methods for the capture of human movement leading to markerless motion capture for biomechanical applications”. In: *J NeuroEngineering Rehabil* 3.6 (2006), p. 3.

[23] T. Oka, N. Komura, and A. Namiki. “Ball juggling robot system controlled by high-speed vision”. In: *2017 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. 2017, pp. 91–96.

[24] *OpenCV.* URL: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga7a6c4e032c97f03ba747966e6ad862 (visited on 11/01/2020).

[25] *OpenCV.* URL: https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#ga69f2545a8b62a6b0fc2ee060dc30559d (visited on 11/01/2020).

[26] A. A. Rizzi, L. L. Whitcomb, and D. E. Koditschek. “Distributed real-time control of a spatial robot juggler”. In: *Computer* 25.5 (1992), pp. 12–24.

[27] *ROSSerial.* URL: http://wiki.ros.org/rosserial (visited on 12/10/2020).

[28] K. Saxena et al. “A study on human-robot collaboration for table-setting task”. In: *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2017, pp. 183–188. DOI: 10.1109/ROBIO.2017.8324415.

[29] Z. Shareef et al. “Dynamical model of ball juggling Delta robots using reflection laws”. In: *2013 16th International Conference on Advanced Robotics (ICAR)*. 2013, pp. 1–8.

[30] C. Smith, M. Bratt, and H. I. Christensen. “Teleoperation for a Ballcatching Task with Significant Dynamics”. In: *Neural Networks* 21 (2015), p. 617.

[31] C. L. Steffi et al. “A Review of the Evolution of Vision-Based Motion Analysis and the Integration of Advanced Computer Vision Methods Towards Developing a Markerless System”. In: *Sports Medicine - Open* 4.24 (2018), pp. 3–4.

[32] *Stereo Calibration and Rectification.* URL: https://medium.com/@aliyasineser/the-depth-i-stereo-calibration-and-rectification-24da7b0fb1e0 (visited on 11/15/2020).

[33] *The Human Memory.* URL: https://human-memory.net/left-and-right-hemisphere-of-the-brain/ (visited on 11/01/2020).

[34] *The Library of Juggling.* 2020. URL: https://www.libraryofjuggling.com/ (visited on 05/21/2020).

[35] Unknown. “Electric Grippers”. In: *How To Choose The Right End Effector For Your Application*, pp. 3–4.

[36] L. Yu and H. H. Ammar. “Analysis of real-time distributed systems: a case study (of robot juggling system)”. In: *[1992] Proceedings of the 35th Midwest Symposium on Circuits and Systems*. 1992, 496–499 vol.1.

[37] H.K Yuen et al. “Comparative study of Hough Transform methods for circle finding”. In: 8.1 (1990), pp. 71–72.

[38] C. Zhang and J. Yang. “Dynamics of Elastic Robots”. In: *A History of Mechanical Engineering*. 2020, p. 484.