# TIF160

## Humanoid Robotics

# *Robotic Manipulation using Linear Genetic Programming and Visual Feedback*

Final Report
October 30, 2020


Chalmers University of Technology
Department of Mechanics And Maritime Sciences
October 30, 2020

*Project members*
Aren Moosakhanian
Jan Paul Theune
Jan Schiffeler
Sourab Bapu Sridhar
**Examiner**
Krister Wolff

# Abstract

In the field of Humanoid Robotics, researchers strive to create robots that are human-like in their behavior, their morphology, or both. Two of the biggest challenges that researchers face today are teaching robots to operate in unmapped environments and make robots come to their own decisions. Deep Reinforcement Learning (Deep RL) and Linear Genetic Programming have emerged as a promising method for developing such control policies autonomously. This project aims to implement a working prototype humanoid robot that uses Linear Genetic Programming to learn to independently pick up objects. The robot can evaluate its performance using computer vision and improve its learning algorithm based on the feedback. The learning algorithm was trained virtually to enhance the first iterations' performance but transitioned to real-world training later in the project.

# Contents

# 1 Introduction

## 1.1 Background

In the field of Humanoid Robotics, researchers strive to create robots that are human-like in their behavior, their morphology, or both. Two of the biggest challenges that researchers face today are teaching robots to operate in unmapped environments and make robots come to their own decisions. This project aims to implement a humanoid robotic system that combines such features. It will result in a working prototype robot that can learn to pick up objects on its own. Today's robots perform very well on repetitive tasks that require precise work in a carefully controlled and monitored environment. However, it is still challenging for a robot to successfully operate in unknown environments by making sense of its surroundings, plan its actions and execute those plans using some means of actuation while monitoring a feedback loop to ensure everything proceeds accordingly. This project aims to help overcome such challenges by building an adaptive control algorithm.

## 1.2 Purpose

This project's main idea was to research the possibilities of creating a real-world robotic movement based on Reinforcement Learning (RL). The idea was later changed to using Linear Genetic Programming (LGP) because of technical difficulties. For demonstration, a robot that learns to pick up objects on its own will be developed. The initial learning will occur in a simulation environment, and the learning would then be transferred to a physical robot. The Hubert robot will be used to perform this task. The Hubert robot will detect a predefined object's position, physical orientation, and hand's location using a mono camera setup. After successful detection, the robot then tries to pick up the object. Finally, the system evaluates the executed actions by predefined measures to determine its performance.

This project will help understand a humanoid robot's interaction with its environment in an unmapped space and help close the reality gap between simulation and the real world.

## 1.3 Aim

After the change from Reinforcement Learning to Linear Genetic Programming, the aim of this project can be summarized as follows:

"To improve our understanding of a robot's interaction with its environment, we will apply the concepts of Computer Vision, Linear Genetic Programming, and applied kinematics to train an existing robot to recognize and pick up an object on its own."

## 1.4 Scope

The scope of this project is to explore the possibility of the Hubert to find the object, recognize its orientation compared to its hand, calculating the optimal change in joint angles needed to pick it up and do so. This project's scope does not include changing Hubert's mechanical structure to improve results or training it for multiple objects in different shapes.

## 1.5 Segments and Boundaries

### 1.5.1 Segments

An overview of the different tasks and boundaries is provided by dividing the project into different sections. These sections are documentation, hardware, computer vision algorithm, linear genetic programming algorithm, and kinematics. The various segments are represented in the table 1 below.

| | |
|---|---|
| Documentation | Planning report (D1) |
| | Final report (D2) |
| | Presentation (D3) |
| Hardware | Set up Hubert robot with one free-moving arm (H1) |
| | Evaluate object to picked up (H2) |
| | Calibrate camera to locate the object (H3) |
| Computer Vision Algorithm | An algorithm which locates the object in the reference frame (V1) |
| | An algorithm to locate the gripper position of Hubert (V2) |
| Linear Genetic Programming Algorithm | Alter an existing simulation model environment for running the linear genetic programming (L1) |
| | An algorithm to learn different possible movements (L2) |
| | An algorithm to pick up objects based on its position (L3) |
| | Transfer learning from simulation to real hardware (L4) |
| Implementation | A low-level driver to activate the motors (I1) |
| | A low-level driver for gripping and picking up an object (I2) |
| Evaluation | Evaluate the performance of the grabbing maneuver (E1) |

Table 1: Segments and deliverables

| Segment | Deliverable | Customer* | Goal |
|---|---|---|---|
| Documentation | D1 | CD | Approval |
| | D2 | CD & TM | Approval |
| | D3 | CD, TM & CC | Approval |
| Hardware | H1 | TM | The Hubert has been set up correctly |
| | H3 | TM | The object has been chosen |
| | H4 | TM | The camera has been calibrated |
| Computer Vision Algorithm | V1 | TM | The algorithm correctly locates the object and gives it position in the reference frame |
| | V2 | TM | The program correctly locates the hand position and gives the data to the learning algorithm |
| Linear Genetic Programming Algorithm | L1 | TM | The simulation environment is set up correctly to simulate the Hubert robot |
| | L2 | TM | The algorithm learns different possible movements in the simulation environment |
| | L3 | TM | The algorithm learns to pick up object in the simulation environment |
| | L4 | TM | The algorithm is transferred successfully from simulation environment to real hardware |
| Implementation | I1 | TM | The motors are moving into the right direction and at the right speed |
| | I2 | TM | The robot correctly grips the object with the right amount of force |
| Evaluation | E1 | TM | A sufficient measure to put our results into perspective |

Table 2: Customers and verification criterions for the deliverables
*Course directors, course examiner Krister Wolff and course assistant Carl-Johan Hoel (CD), Project team members (TM) and course colleagues (CC)

### 1.5.2 Boundaries

- An additional gripper would be added only if necessary
- As a starting point, the algorithm would be trained to detect and grab only one object. Detecting and grabbing other types of objects would be done if there is sufficient time
- The objective of this project is to evaluate the possibilities of using a learning approach for robotic movement, not to create a solution which performs better than a “classical” solution

# 2 Theory

## 2.1 Reinforcement Learning

"Reinforcement Learning is an approach through which intelligent programs, known as agents, work in a known or unknown environment to adapt and learn based on giving points constantly. The feedback might be positive, also known as rewards, or negative also called punishments. Considering the agents and the environment interaction, we then determine which action to take."(1)

The agent in this project was defined as the robot, and the environment was a small table on which one can put objects for the robot to pick up. The feedback was only positive, and the closer the robot got the object, the better the feedback. In this project, a robot must learn to pick up the object by himself using camera feedback. The program was trained virtually, in the beginning, to save time and later taught using real-world data from the installed camera system. Both Deep Deterministic Policy Gradients (DDPG) and Trust Region Policy Optimization (TRPO) architectures were evaluated to perform the learning.

Real-time training on the physical Hubert would be very time consuming as the speed of its movement would limit it. Furthermore, it would be very strenuous on its servos as it would need to run for days on end for a satisfying result. Therefore, before applying the final algorithm on the physical Hubert, training in a simulation environment was planned. MuJoCo, a physics engine aiming to facilitate research and development in robotics, was intended to be used as a simulation environment. The virtual Hubert would be constructed by altering an already existing virtual robot made by OpenAI (2) to have the same degrees of freedom as Hubert. The final Reinforcement Learning architecture would then be implemented in this simulated agent to train and reach a good estimation on any given position before applying to the real Hubert.

## 2.2 Computer Vision

The computer vision program's goal is to accurately estimate the distance between the gripper of the robot and the object and give an accurate position estimation of the object. For this, the OpenCV-library is used in Python, which uses binary square fiducial markers. These markers are glued onto the robot and the object. With the marker glued, the computer can calculate the markers' distance without using a depth camera." The main benefit of these markers is that a single marker provides enough correspondences (its four corners) to obtain the camera pose. Also, the inner binary codification makes them specially robust, especially the possibility of applying error detection and correction techniques."(3)

# 3 Method

## 3.1 Resources

All the resources needed during the implementation of the project are listed in the table 3 below.

| No. | Resource | Supplier | Estimated Cost (SEK) |
|---|---|---|---|
| 1 | Hubert Robot | Chalmers | - |
| 2 | Mono camera sensor | Chalmers | - |
| 3 | Literature | Chalmers | - |
| 4 | Software | Chalmers | - |
| 5 | Computational Hardware | Chalmers | - |
| 6 | Cubes to grab | TM | - |
| 7 | Calibration plate (Aluminum sheet) | Chalmers | - |
| 8 | Printed calibration pattern | Chalmers Printer | - |
| 9 | Marker to detect Hubert's hand's pose | Chalmers Printer | - |
| 10 | Table to place objects on | Chalmers | - |

Table 3: Resources Needed

## 3.2 Coordinate Systems and Transformations

Keeping track of all our different coordinate systems is of high importance for the correct control of Hubert. These coordinate systems were tracked in the below manner:

First of all, we introduced a world coordinate system (WCS), the center most system seen the figure 3.1. The WCS is used as a common ground representation for all positions. Next, we have pattern coordinate systems, but those outputs are not directly visible to the user. In figure 3.1, they can be seen near the gripper and on the cube. Using the camera coordinate system, which is located inside the camera with $\tilde{z}$ facing forward, we can transform the pattern coordinate systems into the WCS.

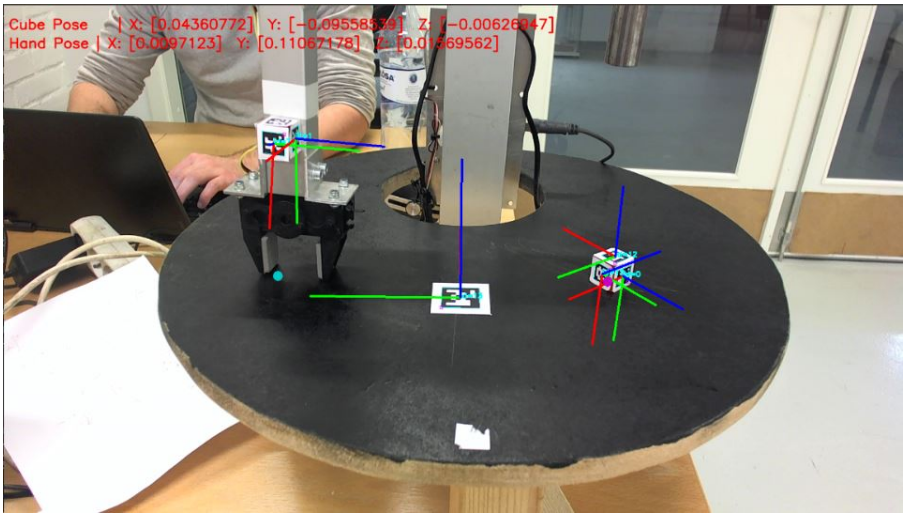


Figure 3.1: Coordinate Systems

There is also a robot coordinate system (RCS) located in Hubert's base and a gripper coordinate system

[GCS] located inside the gripper. Both the coordinate systems can be seen in figure 3.2. However, since the coordinate system in the home problem 1.2 differs from the existing system, some additional transformations were required. In particular, the angle restrictions indicate that a base rotation from 0 to $\pi$ covers the front side. Therefore, the RCS had to be rotated by +90 degrees around $\tilde{z}$. Lastly, the zero-position of the shoulder and elbow motors are different from the ones in homework 1.1. An Arduino program was implemented to perform the required transformation, which converts the GCS angles to the RCS.

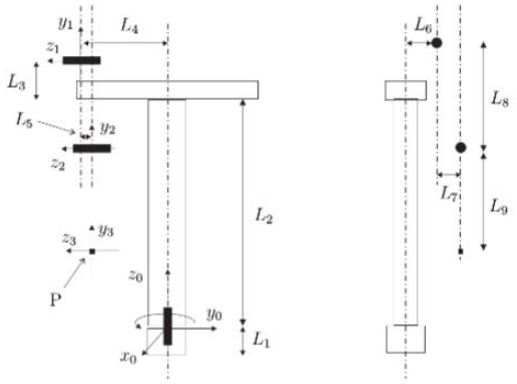


Figure 3.2: Robot System

Note that the RCS is not essential for the project to work. It has only been introduced to build upon the code-base and results of home problem 1.2, but it is imaginable to directly feed all positions in WCS into the learning algorithm.

To finally convert the cube and gripper position, which is detected in world coordinates, into robot coordinates for training, we need to find a transformation matrix from WCS to RCS. Since it is tough to accurately measure the distances and rotations between the WCS and RCS by hand, we use several transformations. First, the cube is placed in the middle of the gripper, measuring its position and rotation with the camera. Then we have the robot move into a predefined position. Since we know the joints' angles, we can calculate the gripper position. Lastly, the transformation from the robot base to the gripper is combined with the measured transformation from the world coordinates to the gripper's pattern. These transformations allow for a simplistic hand-eye-calibration procedure.

## 3.3 Computer Vision

The computer vision program is responsible for detecting both the object to be grabbed (named *cube*) and the gripper's position. The gripper position is only detected because the positional accuracy of the Hubert robot is not very high. In another robot, the absolute position would be calculated base on the input angles.

In general, ArUco markers are detected, which are placed on different objects. These patterns are reliably found in an image, and since they are not symmetric, they provide a sense of rotation. Using multiple patterns per object is done for two reasons. First of all, it needs to be assured that at least one pattern per object is detected to ensure the whole object can be detected. Second, if multiple patterns are detected, that information can be used to refine the position.

Since all patterns have a unique ID, it is straightforward to reference them to their respective object. The more complicated part is to align them to the object's center correctly. The cube is equal for all patterns and only an offset in z direction is needed by half the cube's length. For the gripper, this had to be calculated for every pattern independently.

### 3.3.1 Calibration

To use a standard camera for distance measuring, one has to calibrate the camera first. Calibration means that one needs to find out the camera's intrinsic parameters and distortion coefficients. In this project, ArUco markers are used; with these markers, one can measure the distance between one marker and the camera or two markers. The calibration procedure is similar to that proposed in (4).

The calibration of the camera uses the same library as the actual distance measuring script: OpenCV. The

calibration script uses the function calibrateCameraCharuco() for this. Before one can use this function to calibrate the camera, one needs to print out a Charuco-board, as shown in 3.3.

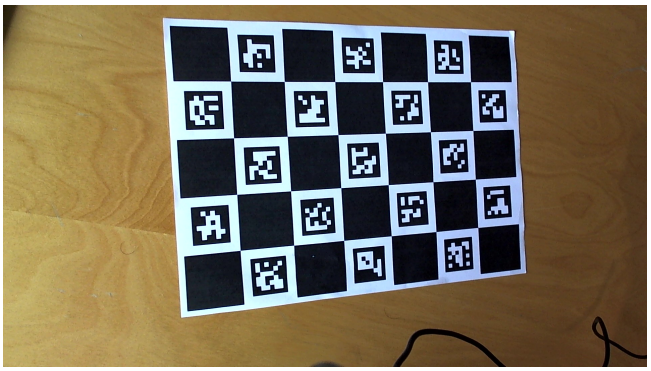

Figure 3.3: Charuco board

Based on an image and the markers' length, the calibration function will store the detected corners and IDs of the markers and calculate the camera matrix and the distortion coefficients. Furthermore, the function can calculate the board's estimated position; however, this data is not used in the calibration.

The camera matrix and distortion coefficients describe the mapping of 3D points to 2D points in an image. With these parameters, one can correct the lens distortion and measure the distance between the markers. These are the numerical values of the camera matrix and the distortion coefficients for the camera in this project:

$$M_c = \begin{bmatrix} 1972.01 & 0 & 806.20 \\ 0 & 1940.8 & 658.99 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$C_d = \begin{bmatrix} -0.169 & 4.805 & 0.00563 & -0.0251 & -33.170 \end{bmatrix} \tag{2}$$

## 3.4 Linear Genetic Programming

### 3.4.1 Initialization

The population was initialized with 80 individuals (chromosomes), all of which have seven genes.

### 3.4.2 Genetic procedures

Three operations were performed each generation to each chromosome.

First, two chromosomes were selected from the population by two independent runs of tournament selection. In tournament selection, four participants were chosen randomly from the population and then made to compete. Competing here means compare fitness values. The chromosome with the highest fitness will likely win, but there is a chance, the worse one is chosen.

Next, these two chromosomes were, with a certain probability, crossed. Crossover means there were two random points chosen in each of the chromosomes. Each chromosome was then sliced into three parts at those points. Lastly, the center parts of both chromosomes were swapped. Since it was not checked that the two center parts' lengths are the same, chromosomes could grow and shrink in size.

The last step was to mutate the chromosomes. There exists an overall mutation probability, but this was adapted to each chromosome's length to have a constant mutation rate per gene. A random number was drawn for each gene in each chromosome, and this gene was altered to another value. The value was checked to fulfill the restrains for a gene at this specific point, such that there was no access to nonexisting register entries or operations.

Additionally, the overall mutation rate started very high and was decreased over time. The change in the mutation rate would enhance exploration initially and allow the chromosomes to converge in later epochs.

Although this wasn't technically a genetic procedure, every round, two copies of the best performing chromosome were kept; this allowed them to mutate more and influence other chromosomes by crossover. This procedure is called elitism.

### 3.4.3 Variables, Constants, Operations

There were seven variables and three constant spaces used in the register. For the constants 1, $-1$, 2 were selected. $\pi$ and 10 were added in some tests but did not seem to be useful and removed to lower the search space.

There were 8 possible operations available to the algorithm:

- +
- -
- *
- /
- arcsin
- arccos
- sin
- cos

The first four operations would use two operands, while the latter would only use the first operand provided. If, when performing /, a division by zero error was encountered, $10^{20}$ was written to the result instead. Also, when the operand for arcsin or arccos was not in the interval $[-1, 1]$, $10^{20}$ was written to the result. Although the approach mentioned above makes less sense than dividing by zero or near zero, which will produce huge results. But it would lead to a bad fitness of the chromosome, and thus it'll be eradicated over time.

During testing, it also seems like sin and cos were not necessary.

### 3.4.4 Data points and fitness function

The data points were created using different angles, and the points were parsed through the forward kinematics. For this, three arrays were created from 0 degree to their respective motor's maximum angle by a step length of 5 degree. Then every combination of those angles was created. Gaussian noise was added before the forward kinematics is calculated to give the data set more variance at each step.

Using a more sparse set of angles at first and increase it once the chromosomes had converged sufficiently was not improving the results. However, pausing the training and giving it a new randomly created set to train on was found to have minor positive effects.

The fitness function f was the inverse of the error e. The fitness function is represented below:

$$e = \frac{1}{n} \sum_{i=0}^{n} \sqrt{(x_i - x_{i-est})^2 + (y_i - y_{i-est})^2 + (z_i - z_{i-est})^2}$$

$$f = \frac{1}{e}$$

n denotes the number of all data points in the set, $x_i$, $y_i$, and $z_i$ were the target points while $x_{i-est}$, $y_{i-est}$, and $z_{i-est}$ were the points suggested by the algorithm. The algorithm's output was the three desired angles, which were then turned back into a location using the forward kinematics model for the error calculation.

Additionally, if a chromosome suggested a solution, the closest possible value inside the respective motor's angle restrictions was chosen. This uncertain procedure will eradicate chromosomes working with this over time.

Approaches with a random creation of checkpoints have been disregarded, as perfect fits to an arbitrary angle were likely and would have produced unusually good results but gotten stuck in the next set. Additionally, the idea of elitism does not work then.

### 3.5 Arduino program

The Arduino program was implemented as an interface to the motors. The program takes the calculated values of the angles from the LGP as a 12 digit number, where every three digits correspond to one position of a motor. The motors used for this project are the body, shoulder, elbow, and gripper motor. The function accounts for each motor's maximum and minimum values and won't move to a position if the program gives out a position that is not possible to go. Furthermore, the motors move continuously; this means that once a position has been set, all the motors will move at the same time to this position.

We also implemented a safety function since the robot operates with a table build around itself. Since hitting the table could damage the motors, there is a maximum height, which the robot can't pass. Each iteration of motor movement checks if the robot is above this height. If the robot would go below this threshold only motor movements which increase the gripper's height can be made. This way, we could ensure that the robot would not get stuck if the trajectory of the gripper to the end position would fall below the minimum high, but the end position was still above.

Lastly, the set-up function has been altered to move the arm to a position from which the robot can easily pick up objects.

# 4 Results

The results of the project are represented as videos. Some screenshot from the videos are attached below:

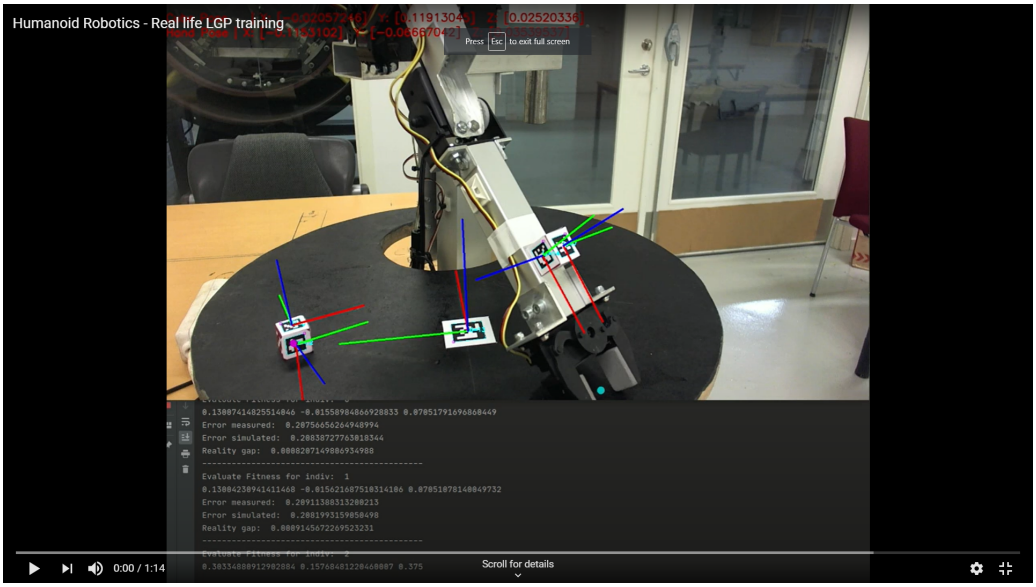


Figure 4.1: Screenshot from video 1

The first video shows the learning algorithm if the program has not been trained previously. As one can see, the arm doesn't grab the object and moves quite far away from the object. However, the camera correctly measures the object's position and gives out the distance between the object and the arm. The dots on the gripper and the object indicate the estimated position from the vision algorithm. The video is posted here

The second video shows the algorithm after virtual training. The camera gives the correct position of the object in the world coordinate frame and the algorithm then calculates a joint angle configuration based on that. Even thought the robot still can't pick up the object, there is a magnificent improvement from the previous video. The video is posted here

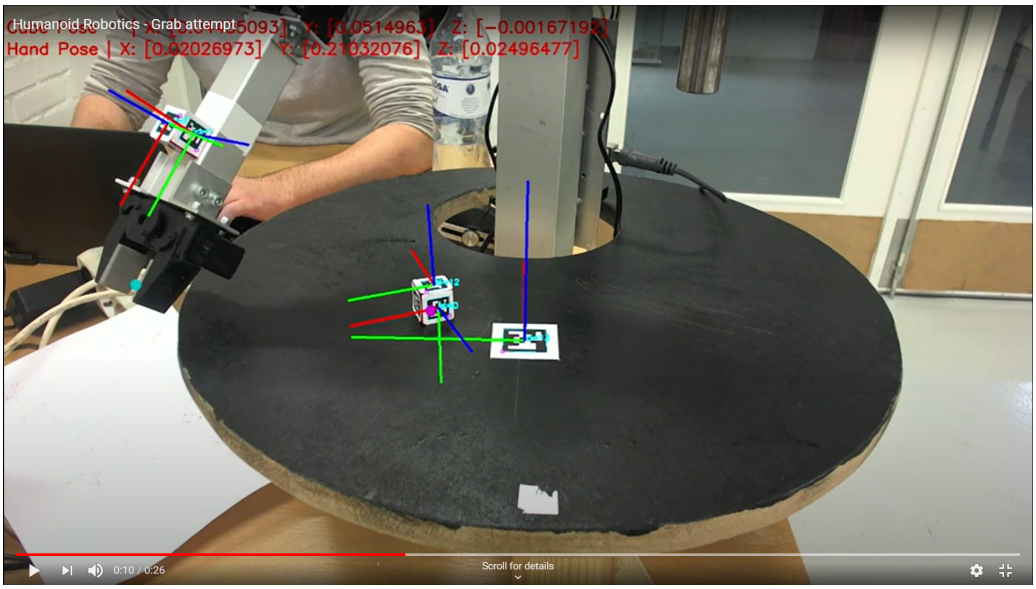


Figure 4.2: Screenshot from video 2

On comparing the results from both the two videos, it can be concluded that the learning algorithm works and can accurately determine the angle configuration based on a given position in the robots reach. However, further training is required for the robot to grab an object as the average distance between the arm and the object is still too high.

Additionally, the computer vision algorithm works well and can detect the object and the arm. As one can see from the vision algorithm's estimation, the estimation is entirely accurate and can correctly map the 3D position from a 2D image.

Lastly, the kinematics part of the project also works well since it accurately moves the motors to the correct angle and does this in a reasonable time frame and in a continuous motion.

## 4.1 Discussion

While the computer vision and kinematics part of the project works quite well, the RL part still needs to be developed further. The simulated learning algorithm cannot give a correct angle composition on average; the algorithm probably needs much more time to learn the inverse kinematic function. Furthermore, the reality gap between virtual training and the real world might be another factor that needs to be addressed.

# 5 Conclusion

This project aimed to improve our understanding of a robot's interaction with its environment. While the training was not completed as we finished the project, we could still implement a good enough solution to show the effects of learning in a real-world environment. We saw that virtual training is helpful when working with this kind of machine learning since it saves much time at the beginning of the training. However, real world training should still be commenced and would further improve the effectiveness of this project.

This project's visual part, which means the location and tracking of an object and the robot arm using binary square fiducial markers, was implemented fully into this project and worked exceptionally well. We were able to track the robot's arm even when it was moving. The angle in which the camera had to see the object was practically irrelevant.

Furthermore, the movement of the individual motors of the robot was also implemented successfully. The robot was able to move freely in its designated space and continuously check if it touched the height threshold in the form of a table and couldn't move past that.

## 5.1 Changes during project

### 5.1.1 Reinforcement Learning (RL)

The learning algorithm initially chosen for this project was Reinforcement Learning, rather than an LGP algorithm. The specific architectures planned for use were Deep Deterministic Policy Gradients (DDPG) at first and Trust Region Policy Optimization (TRPO) later on. However, due to technical difficulties and a lack of supporting documents, the approach had to be modified. If everything had gone as per the plan, the algorithms would be trained in a simulation environment based on the MuJoCo physics engine. After successful training, our idea was to retrain the model in the hardware environment.

## 5.2 Further Work

This project aims to be a starting point from which further research can be commenced. While we were using LGP to train the robot, the idea was to use Reinforcement learning. We hypothesize that Reinforcement Learning would give better results. However, this still needs to be evaluated.

While the camera resolution was high enough, there were some FPS drops when it was running under Ubuntu, which might be responsible for some detection glitches we had. Also, to further improve the detection accuracy of moving objects, it would be advisable to implement some filters, e.g., an extended Kalman filter, to smooth out the motion. Generally, and especially for the cube, switching to another detection approach might be a better idea. The absolute detection precision of ArUco marks can be exceeded by stereo vision approaches that fit a cube into a point cloud (e.g., RANSAC).

Lastly, we think some improvements to Hubert's hardware could be made. The absolute positioning accuracy of Hubert was not very high; this could be improved by attaching sensors to the joints to provide feedback, such as rotary encoders or even simple potentiometers.

# References

[1] A. Nandy and M. Biswas, *Reinforcement Learning*. Berkeley, CA: Apress, 2018.

[2] OpenAI, "Robotics - simulated goal-based tasks for the fetch and shadowhand robots.."

[3] "Opencv: Detection of aruco markers," 29.10.2020.

[4] "Opencv: Calibration with aruco and charuco," 29.10.2020.