UiA University of Agder

Master

Bolts detection and a combination of conventional and reinforcement learning based control of UR5 industrial robot

FREDRIK FRIGSTAD

SUPERVISOR
Ilya Tyapin
CO-Supervisor: Kristian Muri Knausgård

University of Agder, 2021
Faculty of Engineering and Science
Department of Engineering and Sciences

Glossaries

End-effector : The last body on a robot orientated toward the target surface

θ : Joint position angular [radians]

$\dot{\theta}$: Joint velocity [rad/s]

$\ddot{\theta}$: Joint acceleration [rad/s^2]

Simscape: Simulation programe that make it possible to create models of physical systems, and simulate the result.

Euler-angels: Describes the rotation around the x, y, z axis.

Error : A four element matrix describing position of end-effector in the relation to the target position

Index: Scalar from 1 to 100, that is used in waypoint selecting

Targetpose: The target position represented as a 4x4 homogeneous matrix transfer matrix

Target position: The the cartesian coordinates for the target. In this thesis the bolt head.

Acknowledgements

Writing this master thesis has been both challenging and enjoyable at the same time.
A sincere thanks to my supervisors, Ilya Tyapin and CO-Supervisor: Kristian Muri Knausgård for guiding me in the right directions. Without you, there would have been no paper.
I also wish to thank my friends and family for advices and encouragements in the sometimes stressful periods of the writing process.
Hope you enjoy the paper

Abstract

The main objective of this paper is to investigate the possibilities for using reinforcement learning to control a UR-5 robot. The paper also looks at how well reinforcement learning works to control a UR-5 robot.
These questions are answered by constructing of matlab and simulink programes. Based on different mathworks example programs and scripts.
In this study, reinforcement learning only works in the situation it is trained to perform. The author believe that it could work better if it were given other configurations/parameters. This will still be an interesting subject for further studies. According to the research done in this paper, the conventional control have the best control accuracy.

Contents

List of Figures

Chapter 1

Introduction

The university of Agder are in the process of designing a robot that dismantle battery pack on electrical vehicles. This design include a industrial robot that detects bolt heads and moves the robot end-effector to the position of the bolt head. The motivation factor is that it is very fulfilling the be part of a design process that can make electrical vehicles even more environment friendly.

This thesis constructs a simulation model off a UR5-robot that locates screw on a battery pack. The control of the robot will include conventional control and reinforcement learning control. The main goal of the study is to answer how to implement Reinforcement learning control on a industrial robot and how good accuracy it is possible to achieve with the combination off conventional control and Reinforcement learning.

To include reinforcement learning in a robotic-control task is a state of art area off research. Conventional control loop with PI control loop will also be a part of the control of the robot.The conventional PI control implementation will be used without modifications, so it will not be discussed in this paper.

A matlab script that locate the bolt head is to be constructed. And a model off the UR-5 robot will be implemented using the simscape environment in simulink. The researcher will take advantage off mathworks library for example program and files.

The work in this thesis will be limited to bolt location in 2 dimensions, not in 3 dimensions which is required if it should be used on a electrical vehicle battery pack. The main portion of the thesis will be to implement a combination of conventional control and reinforcement control to control industrial robot.

Chapter 2

Theory

2.1 Degrees of freedom

A rigid body have in total 6 degrees of freedom, if no constraints are present. Degrees off freedom can be calculated as the sum off freedoms bodies minus the number off the independent constraints acting on the body. [2]

N= Number of bodies, including ground
J = Number of joints
m= 6 for spatial bodies , 3 for planar bodies
f_i = Number of freedoms provided by joint i

$$dof = m(N - 1 - J) + \sum_{i=1}^{J} f_i \tag{2.1}$$

Calculating degrees of freedom for UR5 robot.:

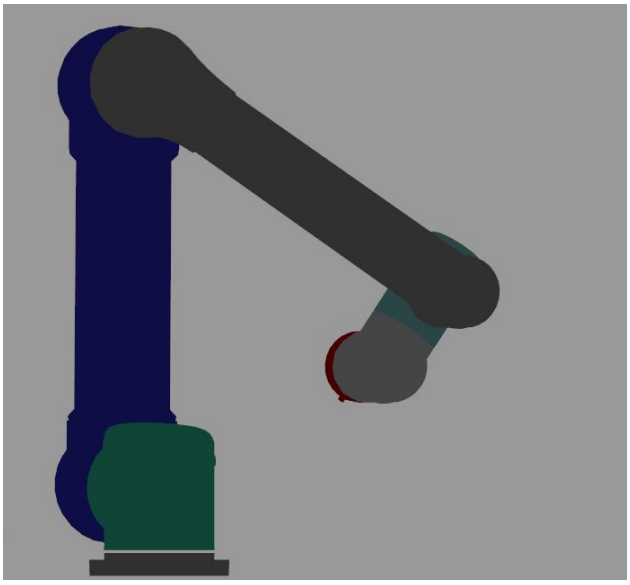


Figure 2.1: UR5 Robot

$$N = 7$$

$$J = 6$$

$$m = 6$$

$$f_i = 1$$

$$dof = 6 \cdot (7 - 1 - 6) + \sum_{i=1}^{6} 1 = 6 \tag{2.2}$$

2.2 Kinematics

Forward kinematics

Forward kinematics describes the position and orientation of the end-effector in relation to the base frame. The end-effector cartesian cordinates (x,y), and the orientation θ can be derived using basic trigonometry. Below is a example for 2R planar open chain.

$$x = L_1 cos\theta_1 + L_2 cos(\theta_1 + \theta_2) \tag{2.3}$$

$$y = L_1 sin\theta_1 + L_2 sin(\theta_1 + \theta_2) \tag{2.4}$$

$$\phi = \theta_1 + \theta_2 \tag{2.5}$$

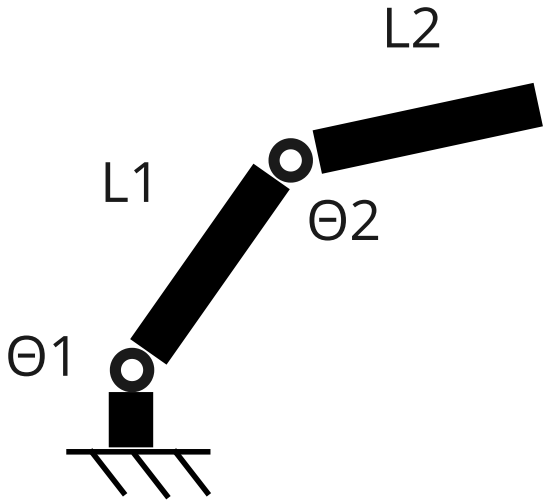


Figure 2.2: 2R planar open chain

A more systematic method for deriving the forward kinematics is to use homogeneous transformations matrices. [3]

Inverse kinematics

Inverse kinematics of a robot refers to the joint position(configuration) that result in a given end-effector position and orientation. Inverse kinematic is used in robotic control in simulink. The inverse kinematic block figure:2.3 converts the end-effector location and orientation from a 4x4 homogeneous transform matrix to the the configurations θ for each joint that correspond to the position and orientation in the homogeneous transform matrix.

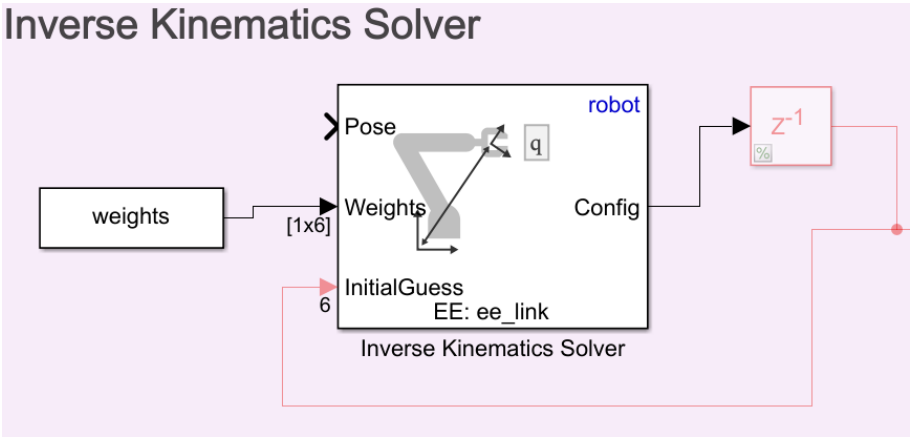


Figure 2.3: Inverse kinematics Block

2.3 Dynamics

Dynamics is the study off what cause the motion off a robot, thus the joint forces and torques. The forward dynamics is used in simulation. ,and describes the angular joint acceleration $\ddot{\theta}$ derived on the angular position θ, angular velocity $\dot{\theta}$, and the torque τ
The inverse dynamics describes the torque τ derived from the angular joint acceleration $\ddot{\theta}$, angular position θ, angular velocity $\dot{\theta}$. Inverse dynamics is used in robotic control

2.4 URDF, Universal Robot Description Format

URDF files contains information about the a robot that is needed for simulations. That include the kinematics and inertia of each body, and link to visualization files like .stl files. Each joint connections is described with one parent link and one child link. Researchers can manually edit the URDF file. If it is desirable to for example add a limit to joint angular position. The URDF can be imported to malab workspace. And then used to generate a simscape model. It can also be used with simulink 3d animation. This reference link to a URDF file for the UR-5 robot.[11]

2.5 Homogeneous Transformation Matrices

Transformation matrix calculate the translation and rotation between a reference frame (x,y,z) and a object frame(x,y,z) Thus the placement off the origin off the object frame in relation to the reference frame. Figure: 2.4

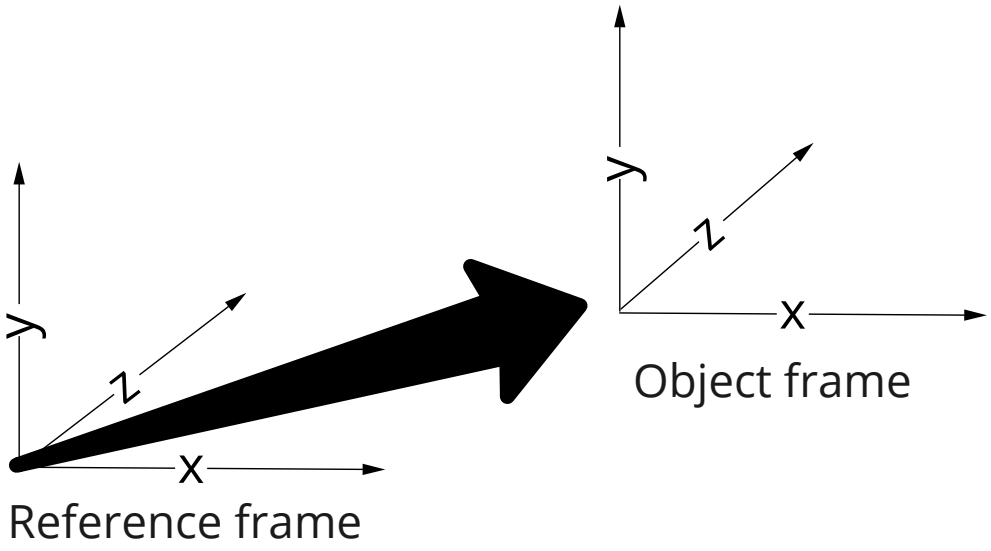


Figure 2.4: Transform from reference frame to object frame

Transformation matrix is represented by 4x4 matrix. That represent the transformation between the reference frame and object frame. [4]

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

The upper left 3x3 off matrix: 2.7 represents the rotation. The 3x1 off matrix: 2.8 to the right represent the translation(x,y,z).

2.6 Reinforcement Learning Concept

Reinforcement learning can be used to replace conventional control loops. A typical RL control consists off a RL-agent that learns based on the input from the environment. The environment is defined as the system/model that agent controls. The agent uses the observations to improve the policy function. The policy function decides value off the outputs(actions). To improve the policy the agent take advantage of the the reward feedback. The reward could be negative or positive. If the actions result in good observations (for example correct wather level in a tank), the reward is positive. Constructing a good reward function is important to make the policy to learn. A state is the result off the actions.

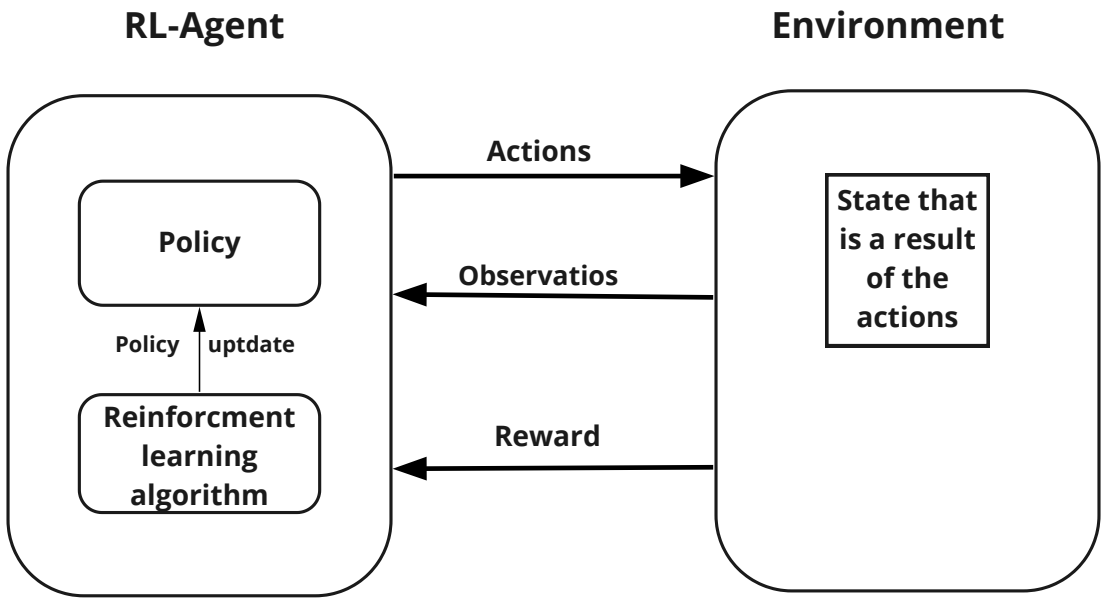


Figure 2.5: RL-Learning concept

Reinforcement learning is a sub type off machine learning. Reinforcement learning works with a dynamic environment that's gives real-time feedback to the agent. Supervised learning and unsupervised learning algorithm not based on real-time feedback. [8]

2.7 Policy

The policy er the par of the agent that decides the action. The policy updates for every episode. One episode is variation off iteration that result in that the episode ends, and the agent either receives a negative or positive reward.

Actor Critic Algorithms

The actor critic algorithm divides the policy function in to 2 neural network, actor and critic. The output from the actor network is the actions. The critic evaluate the actions taken by the actor that result in state. The critic updates the actor for each episode.

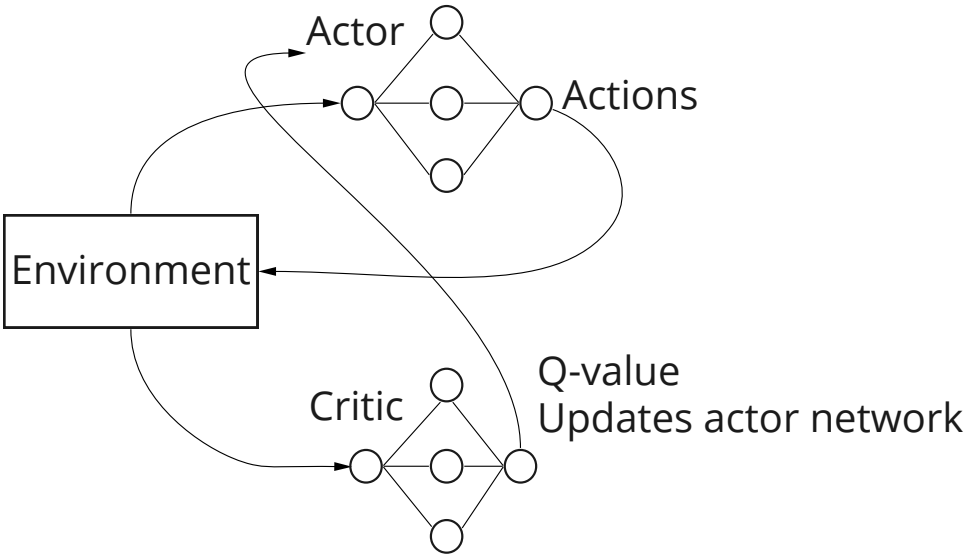


Figure 2.6: Actor critic network

Chapter 3

Simscape/Simulink Model

This mathworks example is used as a base for the robotic simulation. [9] The conventional control sub block that in the referred example, is called "torque control subsystem" see figure 3.1 is unchanged. The measurement of the dynamics is improved. The author have implemented a simscape multi-body model of the robot, sub-block "simscape". Appendix C that is is a modified version of [11] This is done enable implementing of the UR-5 robot and to take advantage of the mechanic explorer (visualisations of robot movement) Further the waypoint generator is changed when implementing reinforcement learning and a torque switch block is constructed. In general the parts that is discussed is changed/improved and the parts not mentioned is unchanged.

The mathworks example [9] also include a matlab script that generates waypoint and loads up URDF file for the robot. This script is modified and a detection and location of bolt head part is added. The bolt detection part are based on [5] The locataion of bolthead part, result in a cartesian cordinates for the bolt head. This cordinates are merged with the excisting waypoint generation. Thus the cordinates are put in as the last waypoint (targetposition) The script that contain the bolt detection and bolt location: A

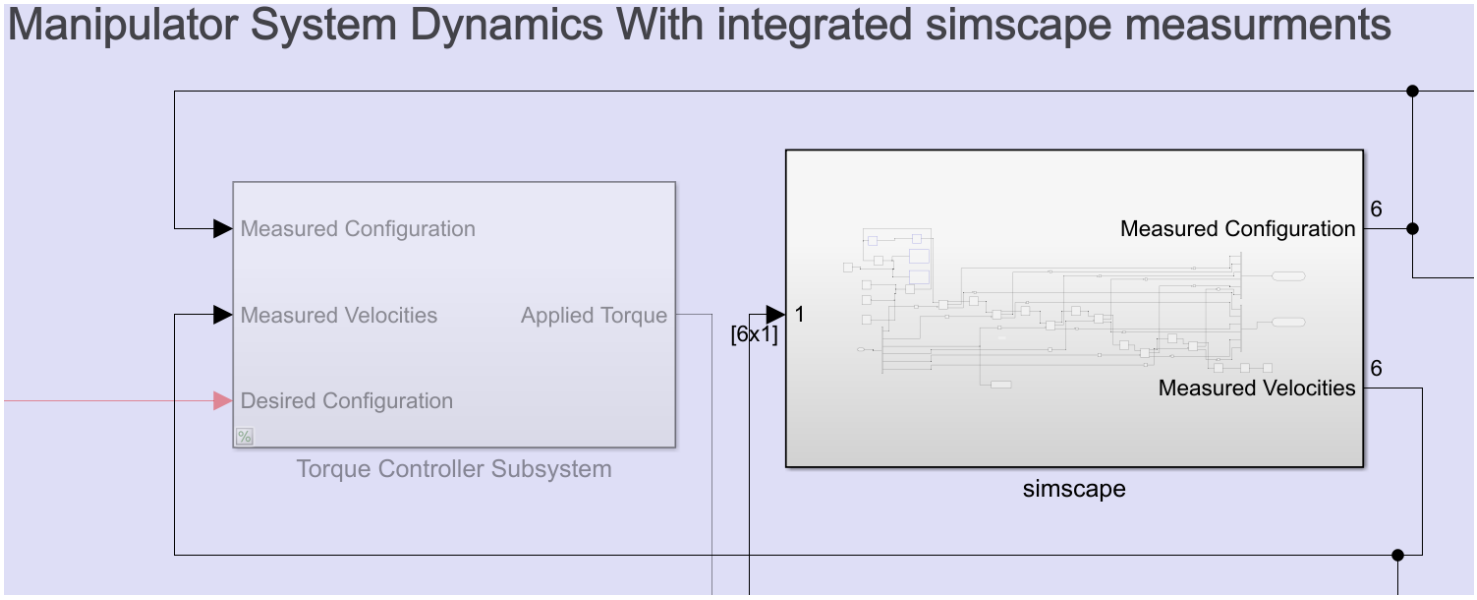


Figure 3.1: Conventional Control Of Joint Torque and Simscape Measurement

3.1 Waypoint Selection

To make the robot move towards the final position, the matlab script A generate 100 waypoints. If the robot is controlled by only the conventional control block. The robot does not move in a strait

line towards the target, and does not reach the target at all.

The waypoints is a 100 points strait line in the space between the robot end-effector initial position, and the target position. The robot end-effector follow this path on the way to reach the target position. Each waypoint consist off a the cartesian coordinates (x,y,z) and the euler-angels :

R_x : Rotation around the x-axis

R_y : Rotation around the y-axis

R_z : Rotation around the z-axis

$$Waypoint = \begin{bmatrix} X & Y & Z & R_x & R_y & R_z \end{bmatrix} \tag{3.1}$$

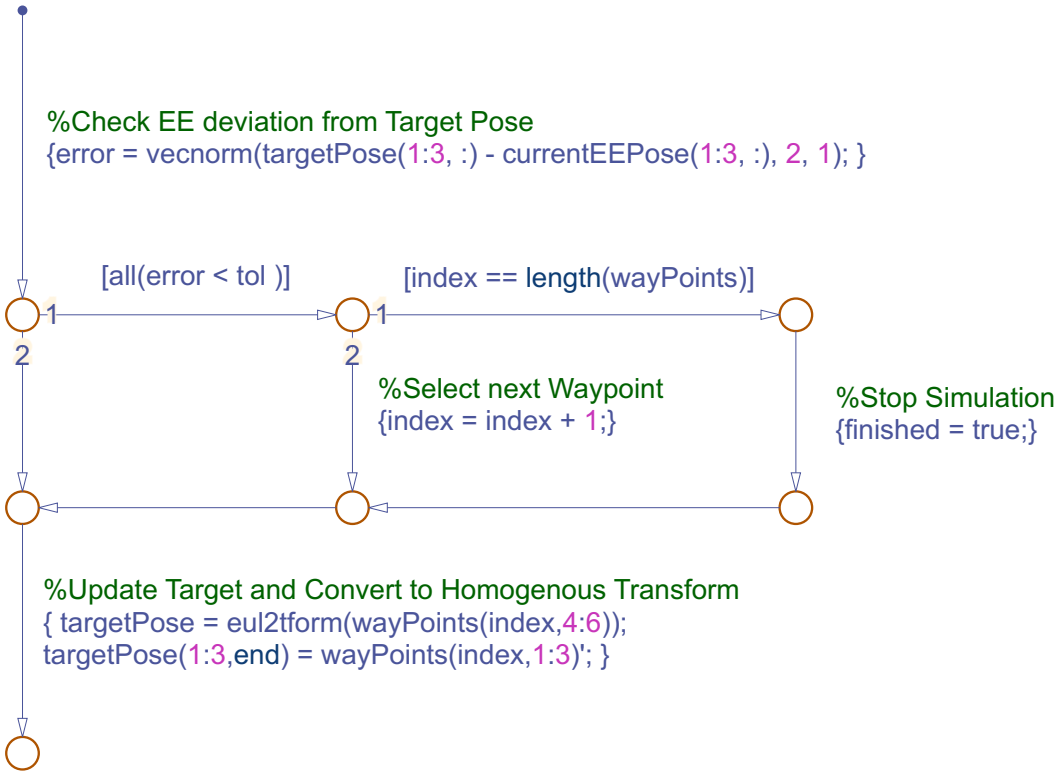


Figure 3.2: Waypoint selection And Generating Error Vector

Figure 3.2 shows the stateflowe block that select the next waypoint if the error is less than a given value (e.g.: 0.01) Inside the stateflow block it is generated an new matrix targetpose Thats converts the waypoint matrix into a homogenous transform matrix, element by element.

$$TargetPose = eul2tform(wayPoints(index, 4:6) \tag{3.2}$$

Transform the euler angulars into 4x4 transform matrix

$$TargetPose(1:3, end) = wayPoints(index, 1:3)' \tag{3.3}$$

Puts the x,y,z coordinates into the last column off the transform matrix. This results in a transform matrix

$$Targetpose = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

3.2 Calculation Of Error Matrix

The error value is calculated as four normalized column vectors. The target position is subracted by the the actual position off the end-effector. The script in Figure 3.2 is:

$$error = vecnorm(TargetPose(1:3,:) - CurrentEEPose(1:3,:), 2, 1)) \tag{3.5}$$

That correspond to:

$$error = TargetPose(3x3) - CurrentEEPose(3x3) \tag{3.6}$$

$$error = \begin{bmatrix} R_{11} & R_{12} & R_{13} & X \\ R_{21} & R_{22} & R_{23} & Y \\ R_{31} & R_{32} & R_{33} & Z \end{bmatrix} \tag{3.7}$$

Then the magnitude of each columns becomes:

$$\|E_1\| = \sqrt{R_{11}^2 + R_{21}^2 + R_{31}^2} \tag{3.8}$$

$$\|E_2\| = \sqrt{R_{12}^2 + R_{22}^2 + R_{32}^2} \tag{3.9}$$

$$\|E_3\| = \sqrt{R_{13}^2 + R_{23}^2 + R_{33}^2} \tag{3.10}$$

$$\|E_{x,y,z}\| = \sqrt{X^2 + Y^2 + Z^2} \tag{3.11}$$

Creating the four element error vector

$$error = \begin{bmatrix} E_1 & E_2 & E_3 & E_{x,y,z} \end{bmatrix} \tag{3.12}$$

3.3 Convential Control Of Robot Workflow

First the matlab script for location of bolt head and generating of waypoints is loaded, appendix A. A is based on the matlabs scripts in [9] This loads URDF file for the UR5 robot in to the matlab workspace. The URDF file is used for the "inverse kinematic block" end the "get transform block". The get transform block converts the measured configuration of the robot into a 4x4 homogenous transform matrix, that is further used in the generation of waypoints. The "location of bolt head" part, locate the target position and saves the cartesian cordinates into the matlab workspace. And is used as the target position for the waypoints.(the last waypoint)

The workflow is described in 3.3 All the yellow block calculations done in the matlab script A the green blocks are part of the simulink programe E. The three red blocks are part of the stateflowe sublock figure: 3.2

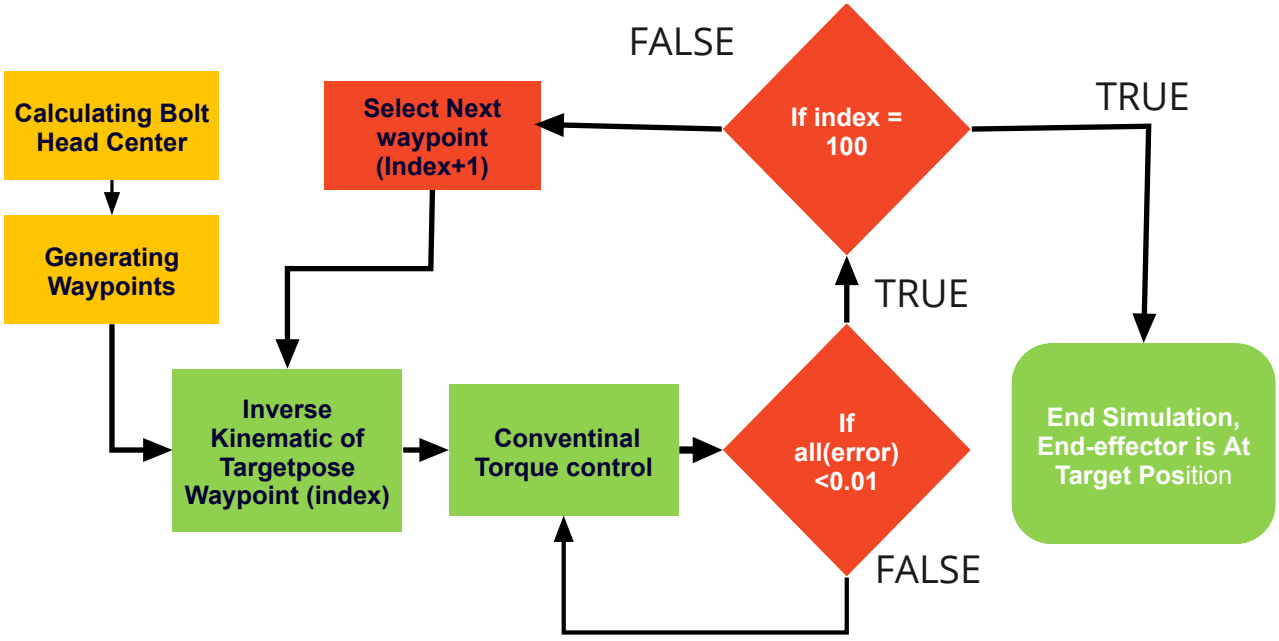


Figure 3.3: Conventional Control Flowchart

3.4 Implementing Reinforcement-learning

The author have chosen to use simulink/simscape to simulate the movement of the robot towards the bolt head. The simulation programs uses both conventional control theory and reinforcement learning to control the robot movements. First the robots actuators are controlled by å PI controller. When the end effector are close to the screw, The RL Agent takes over the control of robots actuators.

This is done by changing the end waypoint in the waypoint selector 3.2 in the Conventional control, and then use the the measured joint configuration in the simulink model for training agent D
The workflow for finding the joint configuration that is used during the RL-training is shown in figure 3.4

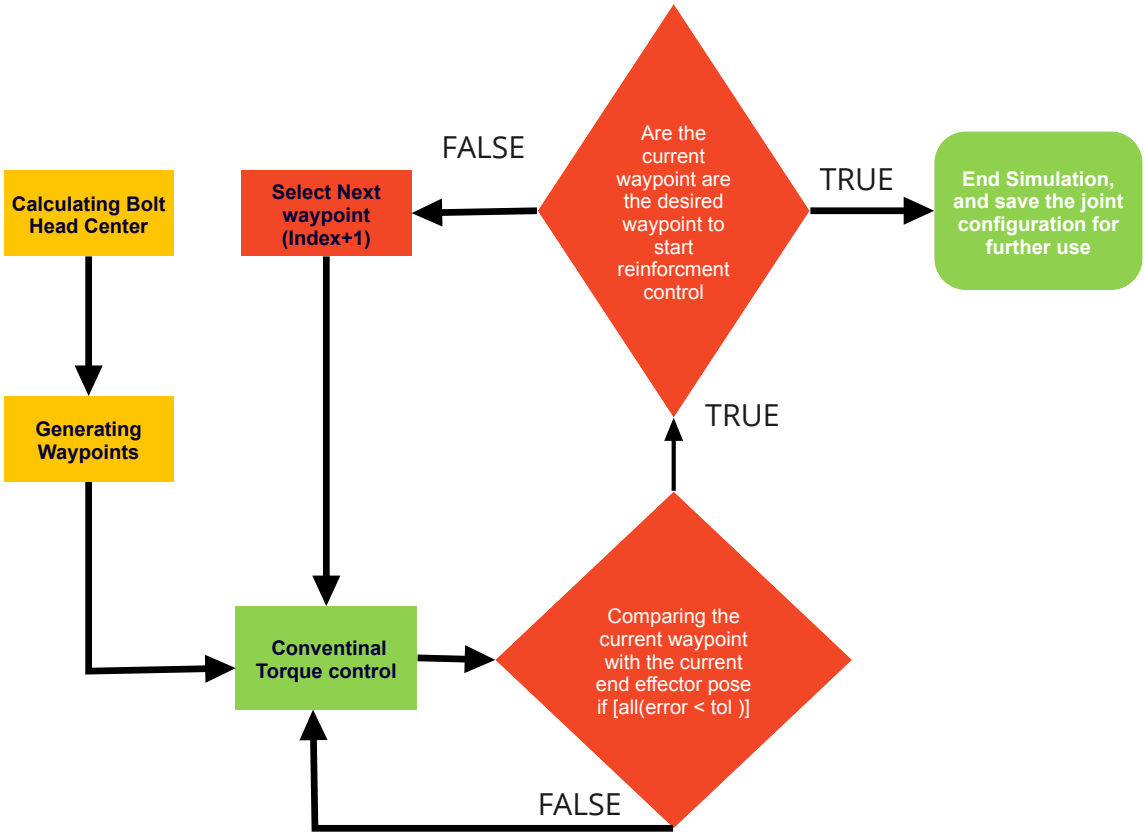


Figure 3.4: Control flowchart for finding joint configurations for further use

3.4.1 Switch Torques To Enable RL-Agent Control

Figure 3.5 Shows the state flow function that enable changing the torque control from conventional control to reinforcement control. When desired waypoint is reach by the conventional control. Switch torque changes to logic "1" this lead to that the output switch the torque output to the "applied torque". "Applied torque that is the result of the reinforcement control.

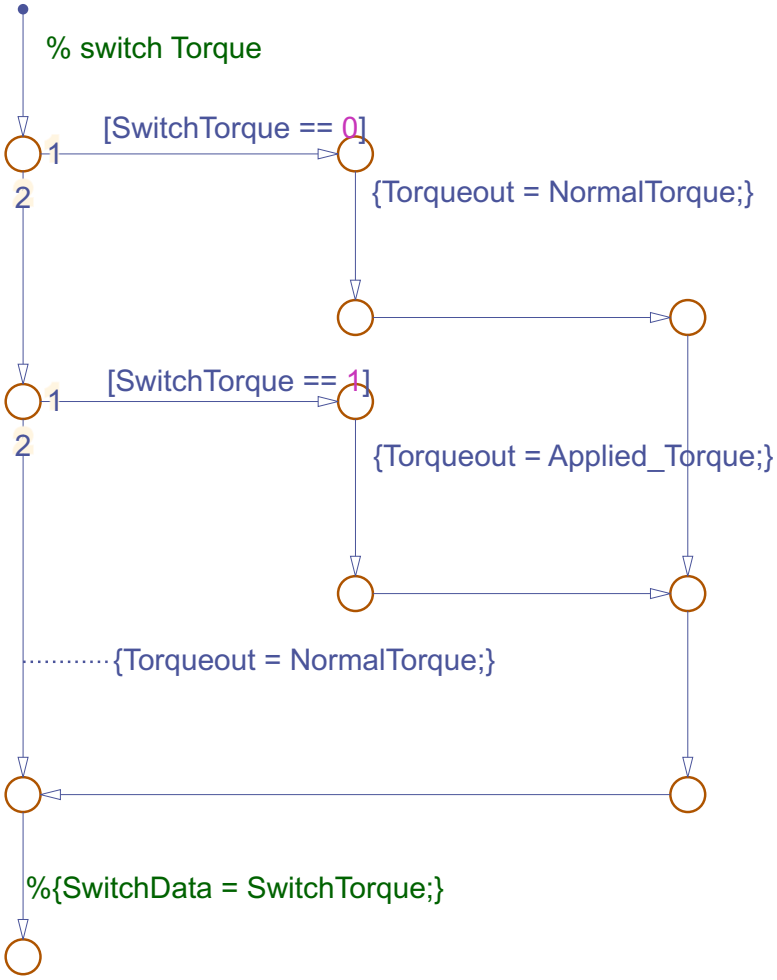


Figure 3.5: Changes Torque

3.4.2 Implementing RL-Agent Control

To control the Robot close to the target-screw the author have chosen to use a RL-agent. [6] this simulink example is used as a base for this implementation. And mathworks have based their work at [10]

Design off RL-agent in Matlab and Creating Enviroment Interface

The script including Rl-agent and enviroment interface is included in appendix B. And this is modified version of the matlab script in [6] . The first part of the script defines the observation interface from the simulink programe "robotsim". The "obsinfo" matrix consist off 12 elements, and the low limit is set to negative infinitive and the upper limit is set to positiv infinitve. The values that is interfaced from the simulink programe are the joint configuration θ and the joint velocity $\dot{\theta}$ from the UR5 robot. It could also include the joint acceleration $\ddot{\theta}$ but this was not included to limit the amount of data to the RL-agent.

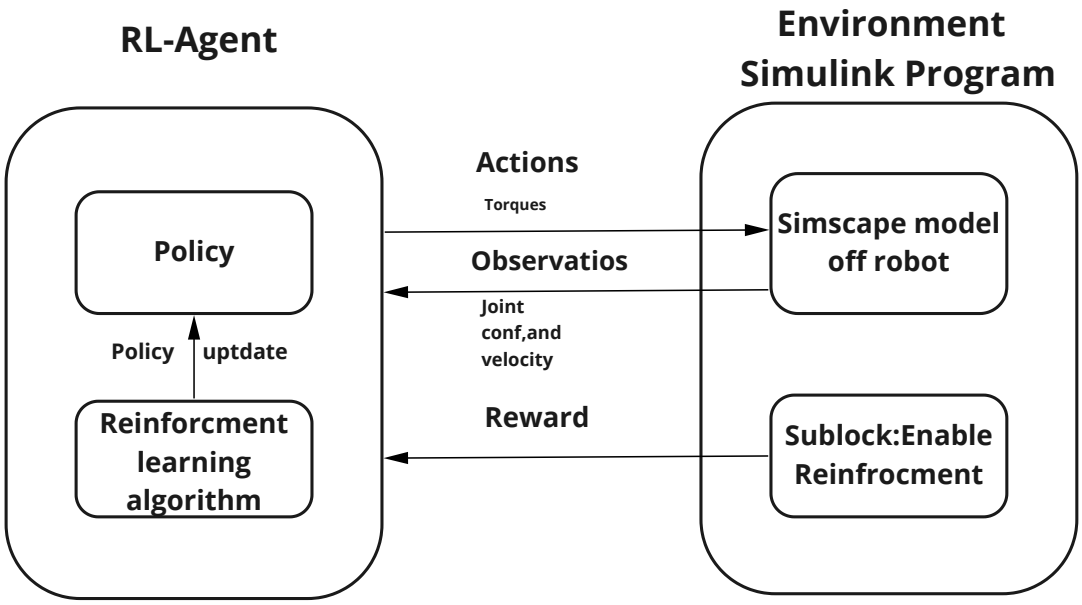


Figure 3.6: RL control

```
open_system('robotsimTrain')
obsInfo = rlNumericSpec([12 1],...
      'LowerLimit',[-inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf ...
         -inf]',...
      'UpperLimit',[ inf inf inf inf inf inf inf inf inf inf inf inf]');
 obsInfo.Name = 'observations';
 obsInfo.Description = 'conf, vel';
```

The action specifications is set to the joint torque τ.

```
actInfo = rlNumericSpec([6 1]);
actInfo.Name = 'Torque';
numActions = actInfo.Dimension(1);
```

The enviroment interface object is located inside the simulink sub block "Enable Reinfrocment"

```
env = rlSimulinkEnv('robotsimTrain','robotsimTrain/EnabledReinforcment/RL ...
    Agent',obsInfo,actInfo);
```

Sub Block Enable Reinforcement

Figure 3.7 shows the simulink block including the RL-agent, "Calculate reward block", and the "Reward block". The "Reward block" end each episode if the error value exceeds the lower or upper limit. Thus when the robots end effector either is close to the target screw, or to far away. Figure 3.8 Exceeding the low limit result in positive reward (distance to bolt head less than 0.5mm). Exceeding the upper limit results in negative reward. Exceeding the lower limit result in positive reward.

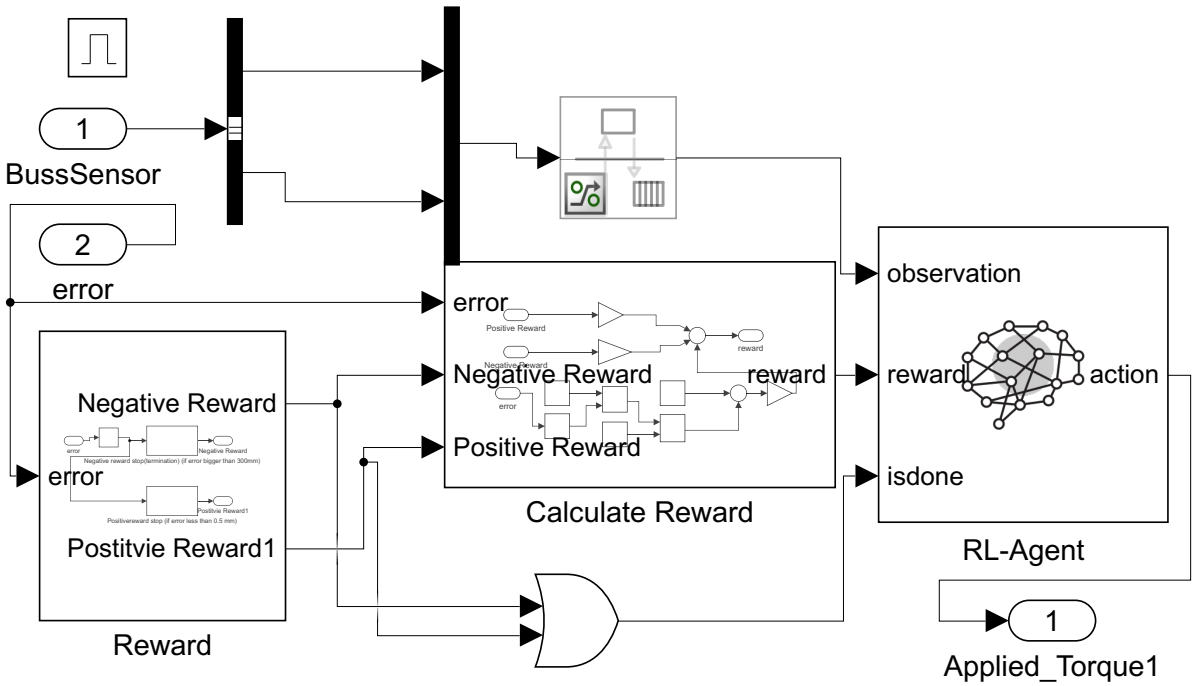


Figure 3.7: Enable Reinforcement Block

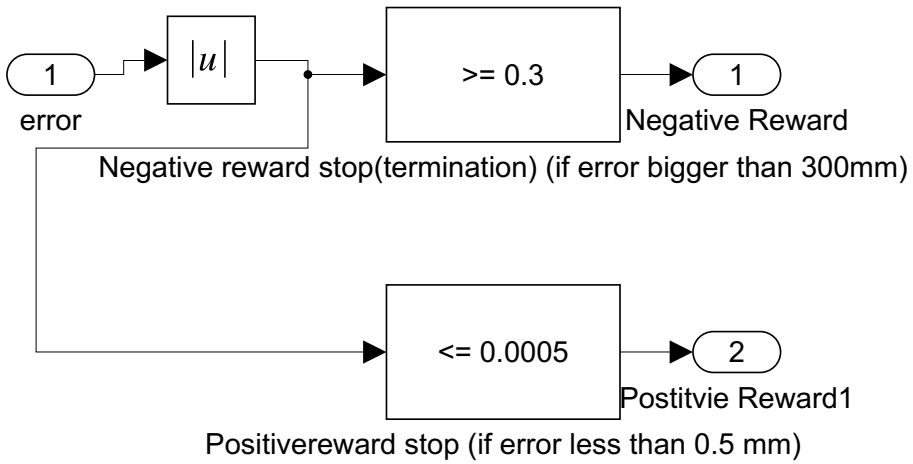


Figure 3.8: Reward Block

Reward function

The author have chosen a reward function that increases exponentially as the error (end effector distance from target) decreases.[1] The reward function can be manipulated to increase the reward value for each episode.

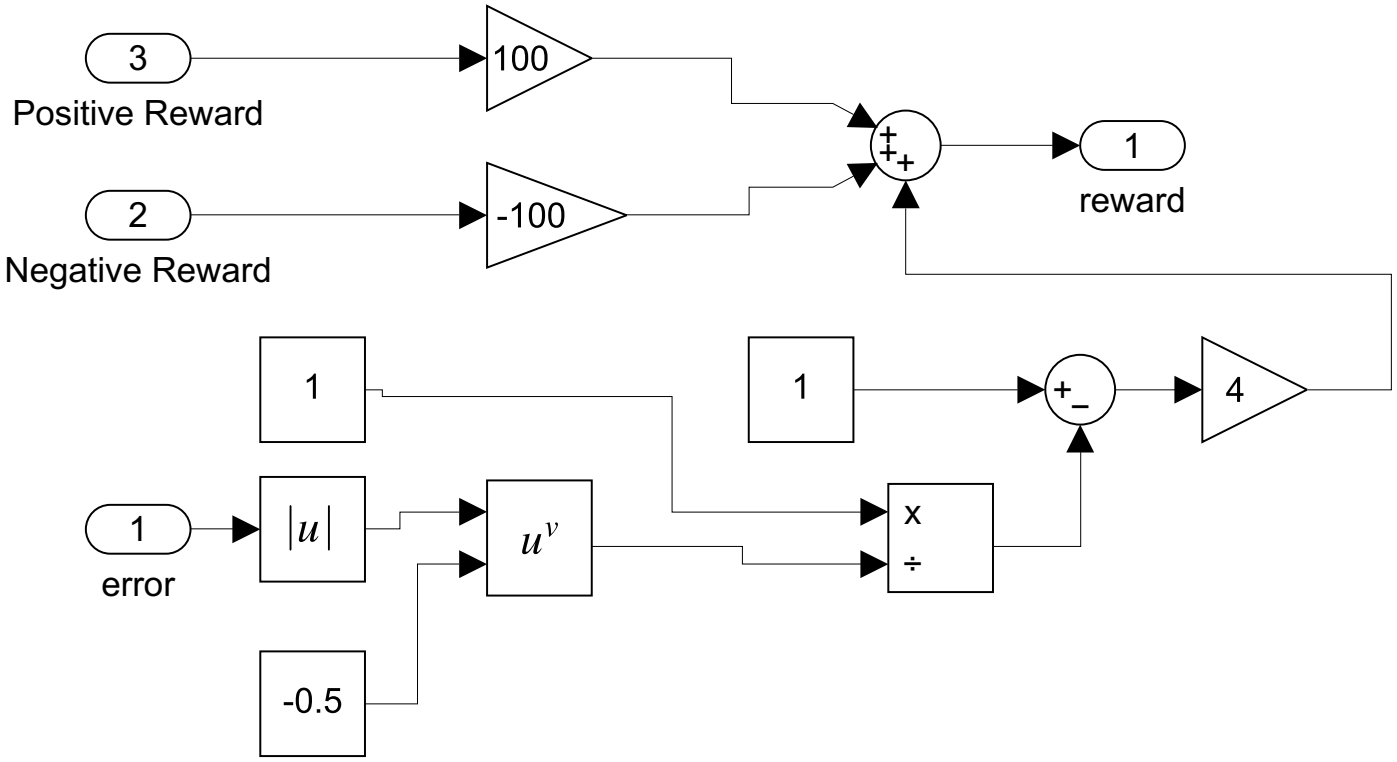


Figure 3.9: Reward Calculation

$$Reward = 1 - \frac{1}{error^{-0.5}} \tag{3.13}$$

$$Reward = 1 - \frac{1}{error^{-0.5}} * 9 \tag{3.14}$$

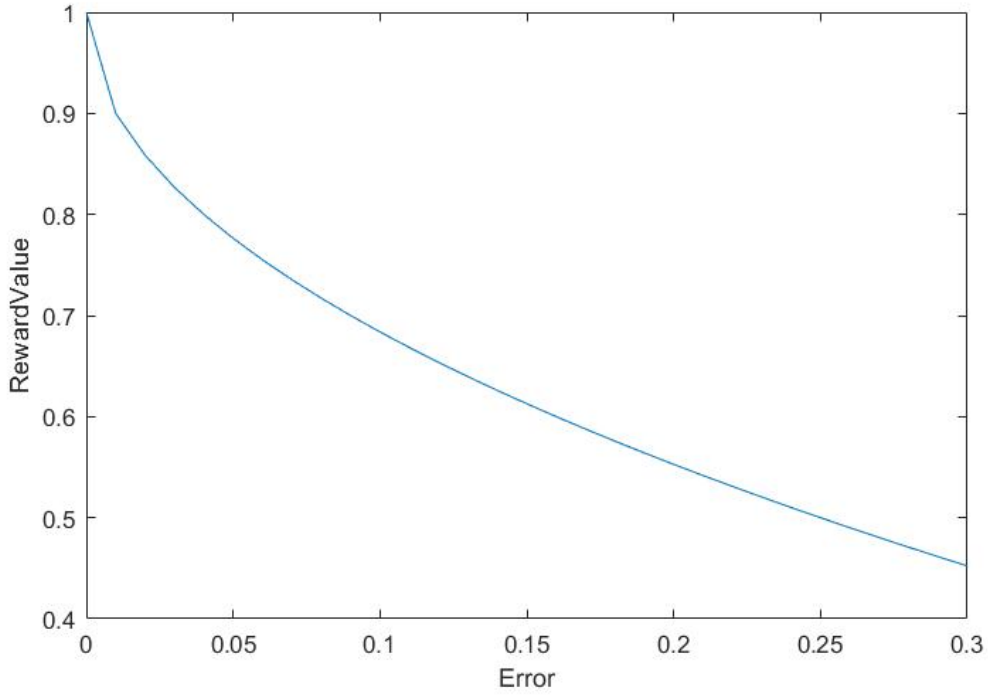


Figure 3.10: Plotted exponentially reward function

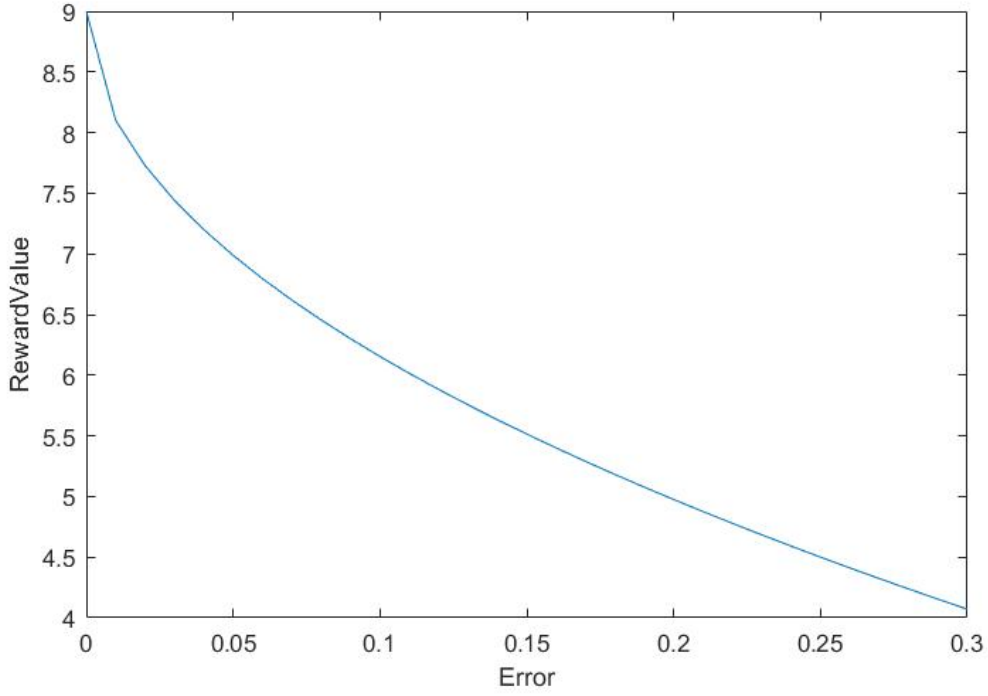
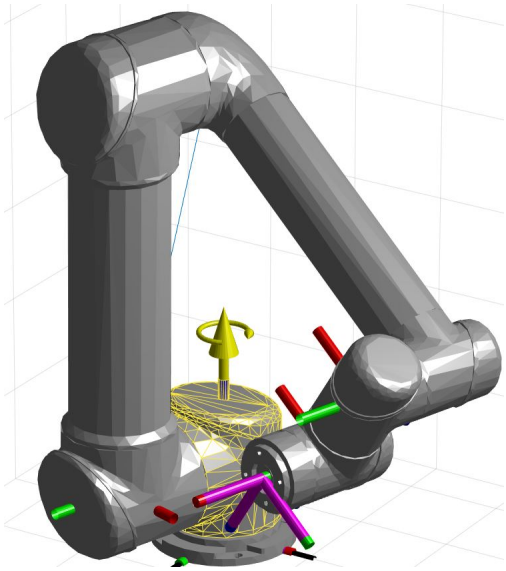


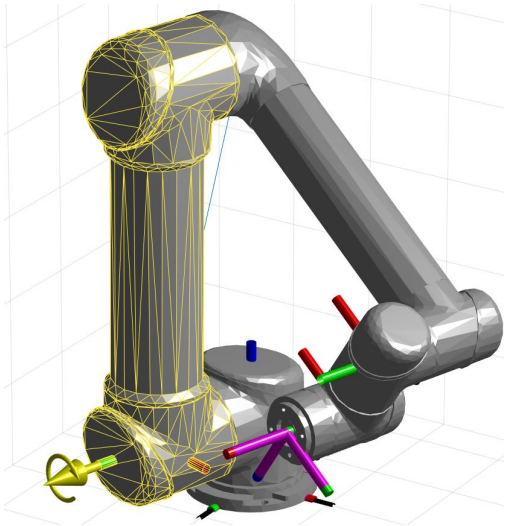
Figure 3.11: More potential exponentially reward function

3.5 Decreas of freedom/model of UR5 Robot

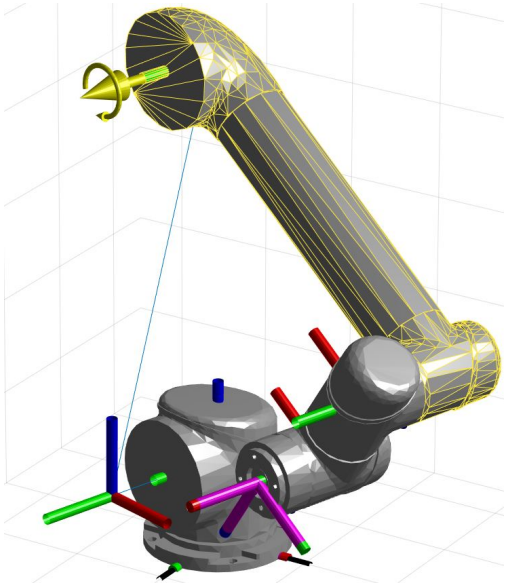
The UR5 robot have 6 degrees of freedom. In figure 3.12 the revolution joint 1-3 are visualized. In figure 3.13 revolute joint 4-6 are visualized. The axes of rotation is indicated with a yellow arrow.



(a)

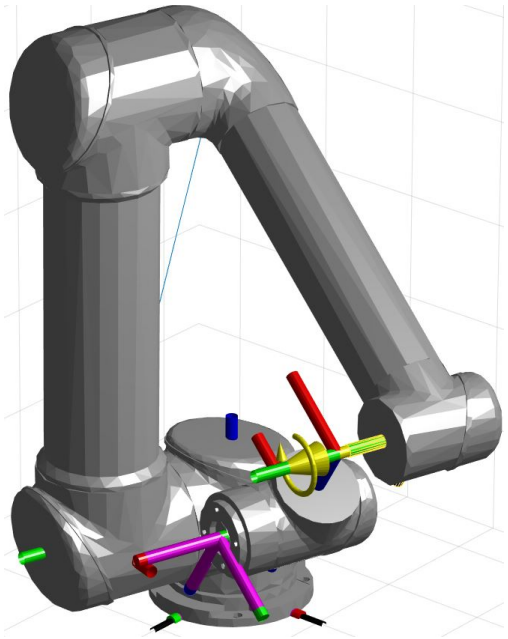


(b)

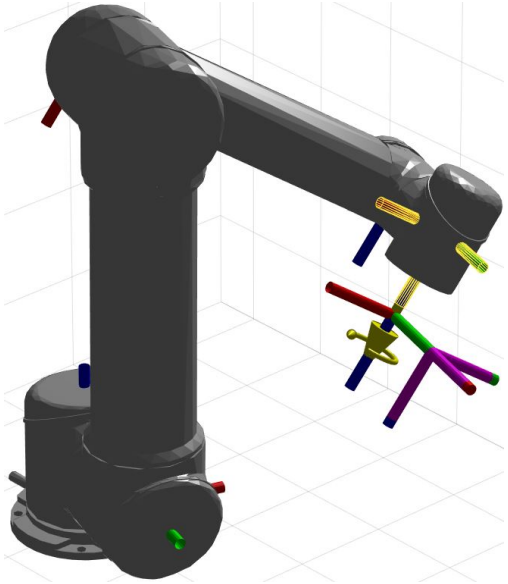


(c)

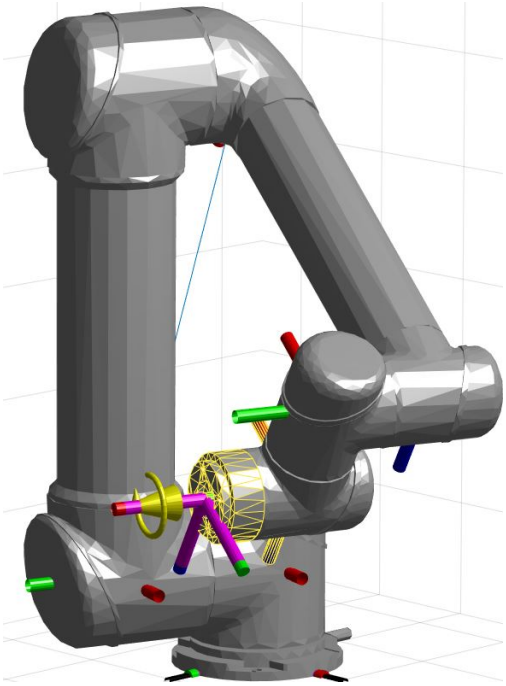
Figure 3.12: Axes of rotation, joint 1-a, joint 2-b, joint 3-c



(a)



(b)



(c)

Figure 3.13: Axes of rotation, joint 4-a,joint 5-b,joint 6-c

3.6 Including Live Camera Stream And Feed-Back To RL- Agent

This idea is to place a camera in the simscape enviromnent and feed the error(distance to bolt direct to the RL-agent. There is now tools for camera implementation in the simscape libary. To do this reasarcher should use simulink 3d simulation.

3.7 Bolt Detection And Location of Bolt Head

This section is partly based on [5], [9] and [7] . The camera is placed 0.7 meter above the battery pack and centered in relation to the robot y-axes. This is illustrated in figure 3.14 The matlab script for location off bolt head is in appendix A

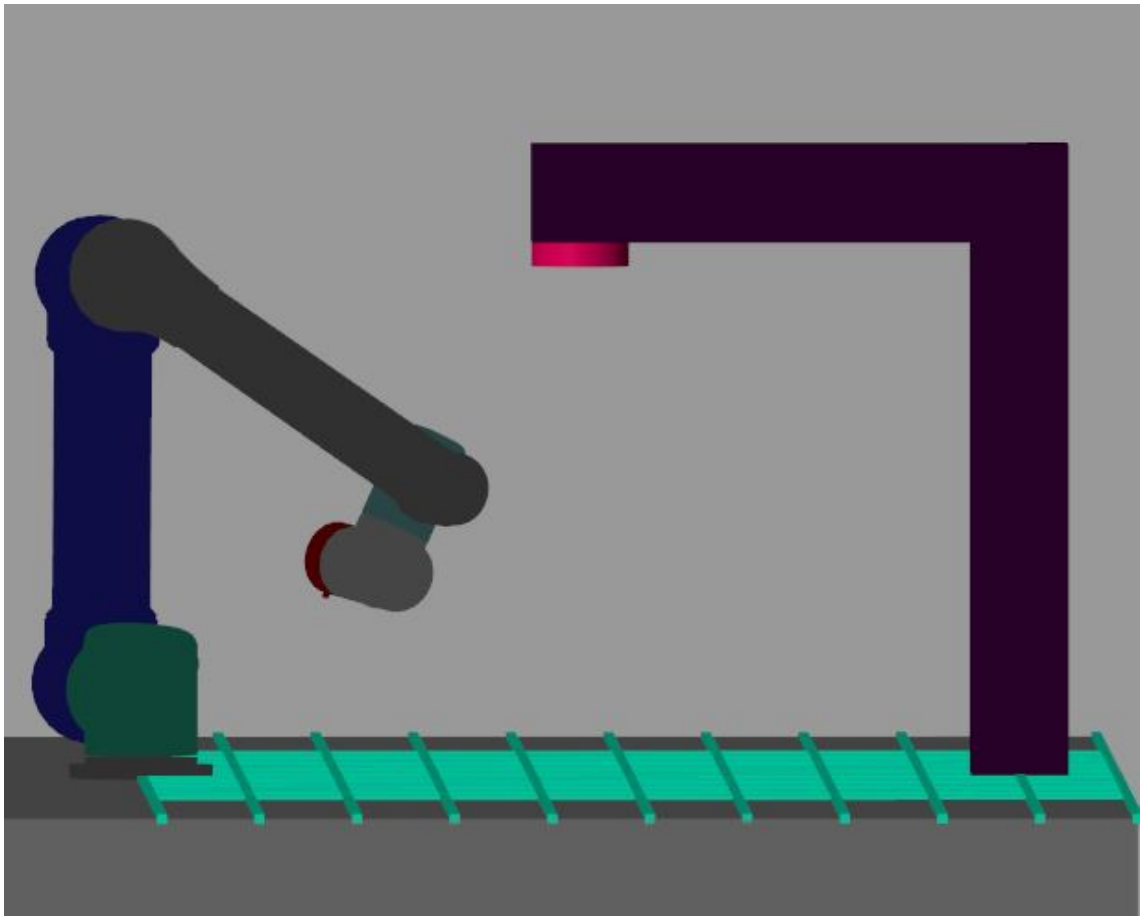
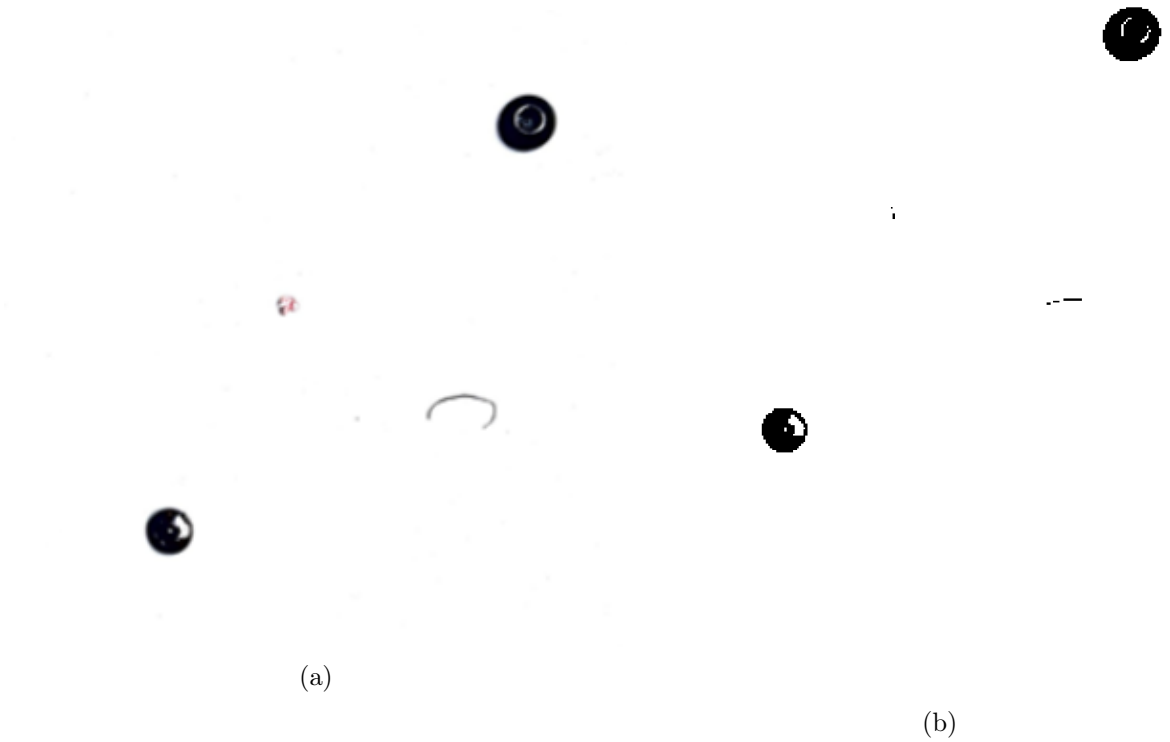


Figure 3.14: Camera placement illustration (red circle)

3.7.1 Matlab Script Workflowe:

- The script first convert the image to a binary image. The matlab function "im2bw" is implemented. All pixels over a threshold (default0.5) get converted to logical "1" (white), the rest off the pixels converts to 0 (black) see figure 3.15a and 3.15b
- Figure 3.16a converting from black and white. This is done to be able to detect the boundary of the object.
- Figure 3.16b Remove information less than a chosen pixel value. The author had to adjust this for this specific picture, so this part could be improved in further research.
- Fill holes (so that the objects is clarified) Figure 3.17a This is done to avoid the "bwboundaries" function to trace boundaries inside the circle (bolt boundaries)

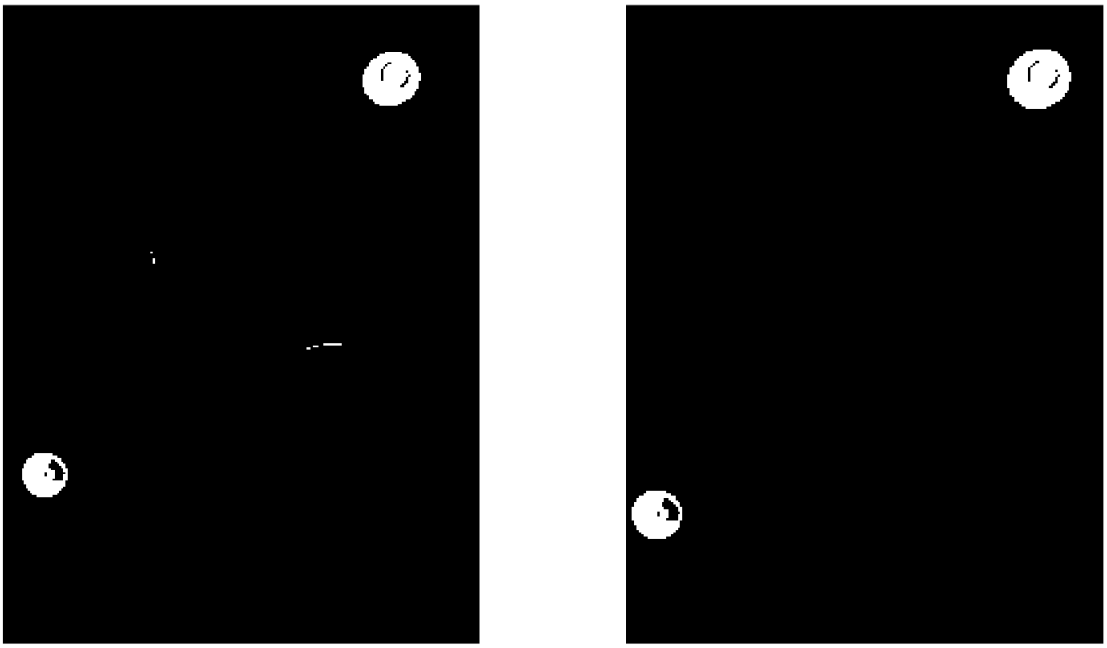
- Trace the boundaries of the objects. Figure 3.17b
- Isolate round objects [7]. Could be implemented, but it was not necessary with this picture.
- Convert the pixels to distance [m]



(a)

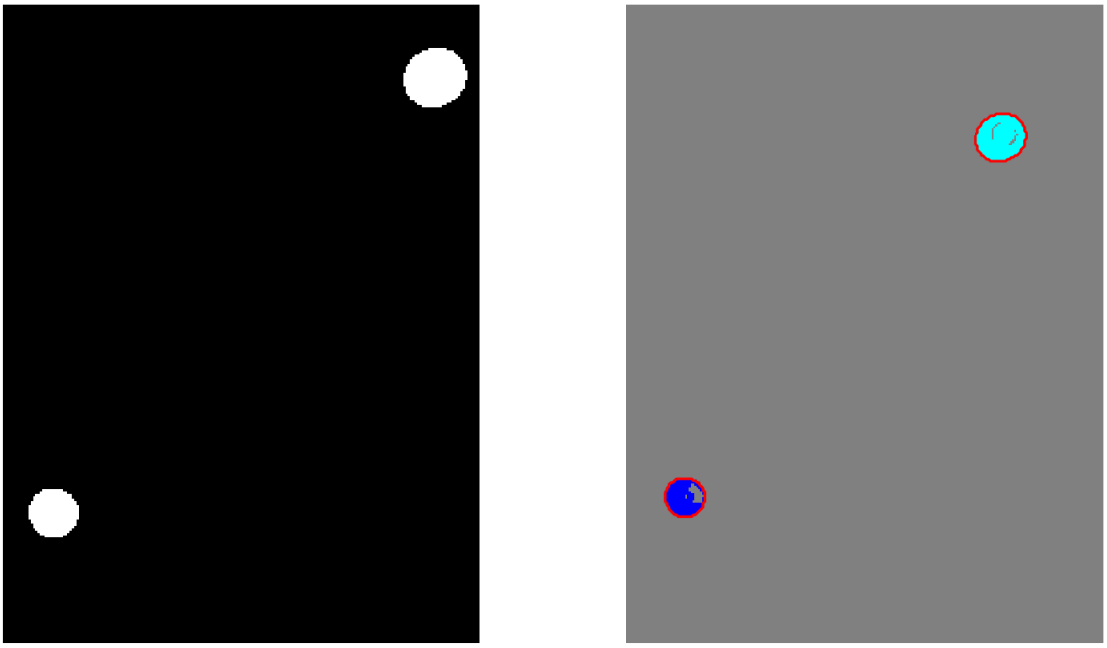
(b)

Figure 3.15: Original image, a and digitized image b



(a) (b)

Figure 3.16: Converted image withe/black figure:a, Reduced noise figure:b



(a) (b)

Figure 3.17: Image fill holes figure: a, Image with boundary detection implemented figure b

From Pixels To Cartesian Coordinates

To convert from pixels to coordinates the script multiplies the digital image with a constant. This constant does not move the origin off the bolt head. It just scales the size picture, See figure 4.2

3.7.2 Calculate The Center Of Each Bolt

The script generates a x-y coordinates for the boundary off each bolt. To find the center off each bolt, the script use the boundary coordinates for each bolt in the x and y direction, thus distance to bolt edge X and distance to bolt edge y. The center for each bolt is located simply with adding the radius off the bolt width the distances to bolt edges. See illustration 3.18

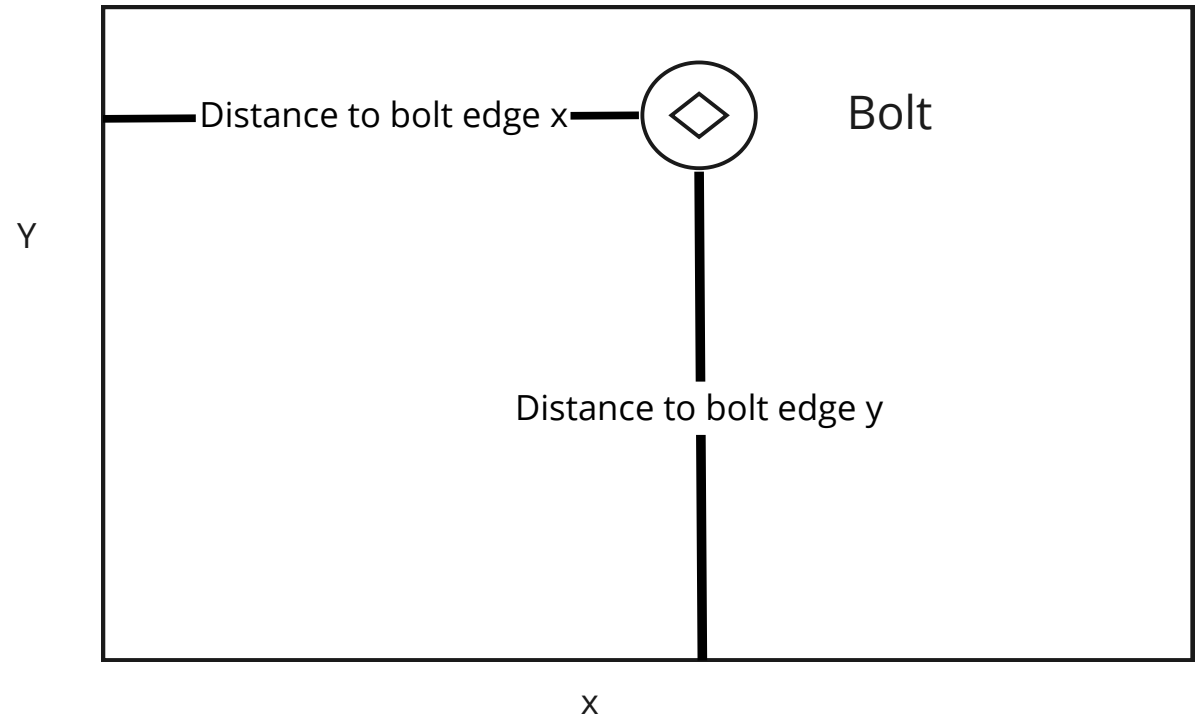


Figure 3.18: Distance to bolt

```
   Bolt1xWidth = max(cord(:,1))-min((cord(:,1))); % max value for x direction-
% bolt 1 width(diameter) = x maxValue - x minValue
% "cord" represents the cordinates for the bolt

   Bolt1xLocation=min(cord(:,1))+Bolt1xWidth/2;
% bolt1 "x" cordinates = start off left bolt edge + bolt radius
% Same approch for the "y" cordinates
   Bolt1yWidth = max(cord(:,2))-min((cord(:,2)));
   Bolt1yLocation=min(cord(:,2))+ Bolt1yWidth/2;
% Put the x and y cordinates for each bolt in a 2 element matrix.
   Bolt1Center(1,1)=Bolt1xLocation;
   Bolt1Center(1,2)=Bolt1yLocation;
```

3.8 Training Agent

As described in Figure 3.4 . The RL-Agent controls the actuation off the robot joints when the end-effector is close to the target screw. To be able to achieve this (to train the agent) the author first runs the simulation with the conventional robot control, and stops the simulation when the end-effector are close to the target screw. Then disable the conventional control block and replace it with the RL control block. This configuration is only for the RL-training purpose only. Simulink program for training agent is in appendix D
A initial target is put in the all the revolute joint off the in the simscape model. Figure 3.19 Shows the measured configuration for each joint. The conventional control block is replaced with the Reinforcement control block . The conventional control runs to waypoint number 93 figure 3.20. At this point the end-effector is 6.2 mm above the target. When the RL-agent take over the control off the robot, the end-effector starts to moves downwards because off gravity. So this is to let the RL-agent control some time and distance to adjust the end-effector to the target position.

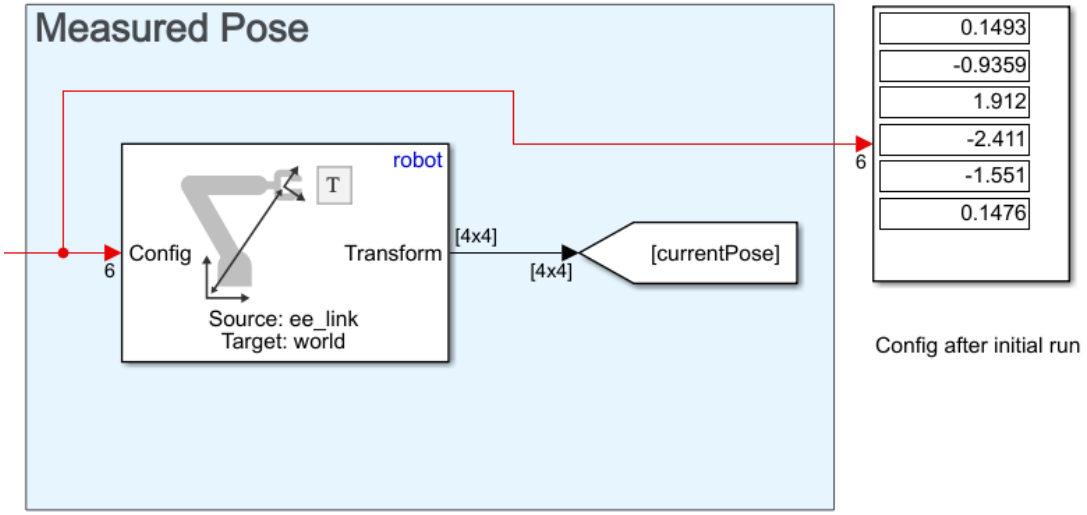


Figure 3.19: Joint config after conventinal control

	X	Y	Z	R_x	R_y	R_z
92	0.5364	0.1916	0.0162	0	1.6977	0
93	0.5401	0.1915	0.0141	0	1.6819	0
94	0.5438	0.1914	0.0121	0	1.6660	0
95	0.5475	0.1913	0.0101	0	1.6501	0
96	0.5512	0.1912	0.0081	0	1.6343	0
97	0.5549	0.1912	0.0061	0	1.6184	0
98	0.5586	0.1911	0.0040	0	1.6025	0
99	0.5623	0.1910	0.0020	0	1.5867	0
100	0.5660	0.1909	0	0	1.5708	0

Figure 3.20: Table showing last waypoints

3.8.1 Waypoint Selection For Training Agent

For training of the reinforcement control the waypoint selector is changed again. The end waypoint is changed to number 100, the target position. See figure 3.21 The error function remains the same as for the conventional control, but a new output "error1" is added. "Error1" extract only the last element of the error matrix 3.15. The function that select next waypoint is removed.

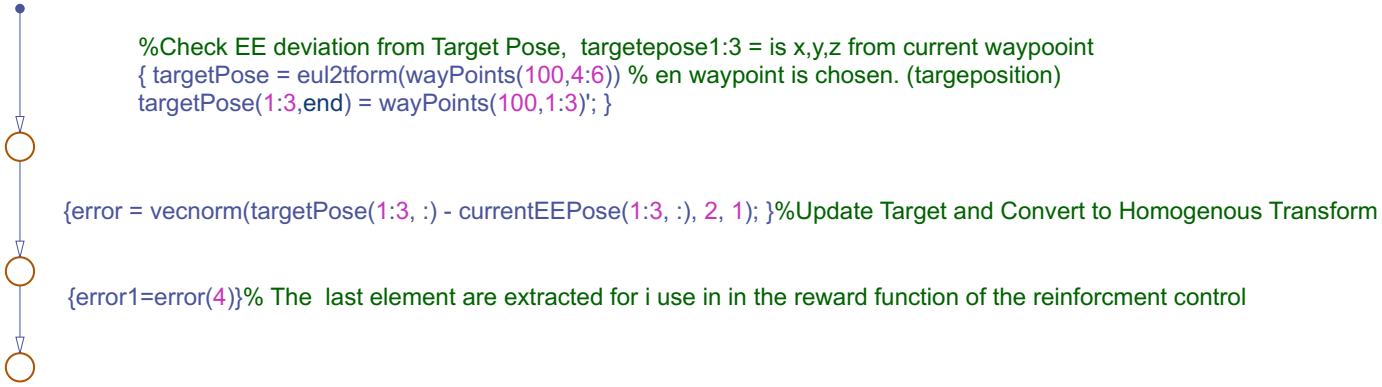


Figure 3.21: Waypoint Selection For Training Agent

3.8.2 Agent Tuning part 1

The error matrix 3.15 function used in this training. The error value is the sum off the off the four elements in the error matrix. The last element in the error matrix describes the distance from the end-effector to the target position. The first 3 elements describes the orientation. The reason for including all four elements instead of just the last element, is to also include the orientation.

Figure 3.22 Shows the result from training the training episodes. There are some episode with reward greater then 400, but the total trending is not not positive. The training is run for total 5000 episode.

The error value for this training session is the sum of the four elements in the error matrix 3.15. The end-effector is at the position for waypoint that correspond to waypoint 92.
Figure 3.23 The upper limit is set to 0.2 m and the the lower is set to 0.001. Figure 3.24 Shows the reward values.

$$error = \begin{bmatrix} E_1 & E_2 & E_3 & E_{x,y,z} \end{bmatrix} \tag{3.15}$$

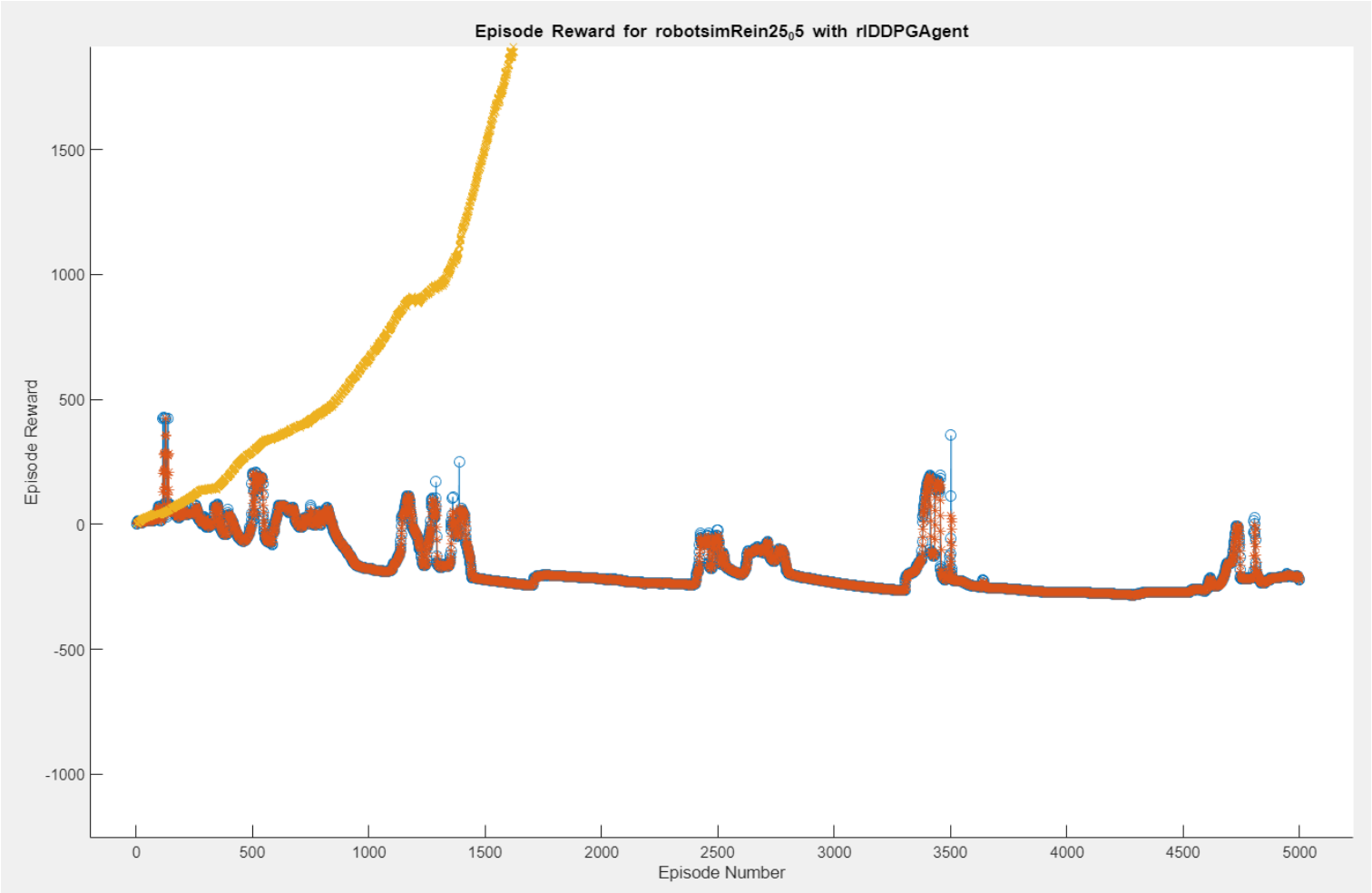


Figure 3.22: fig:Training 1 results

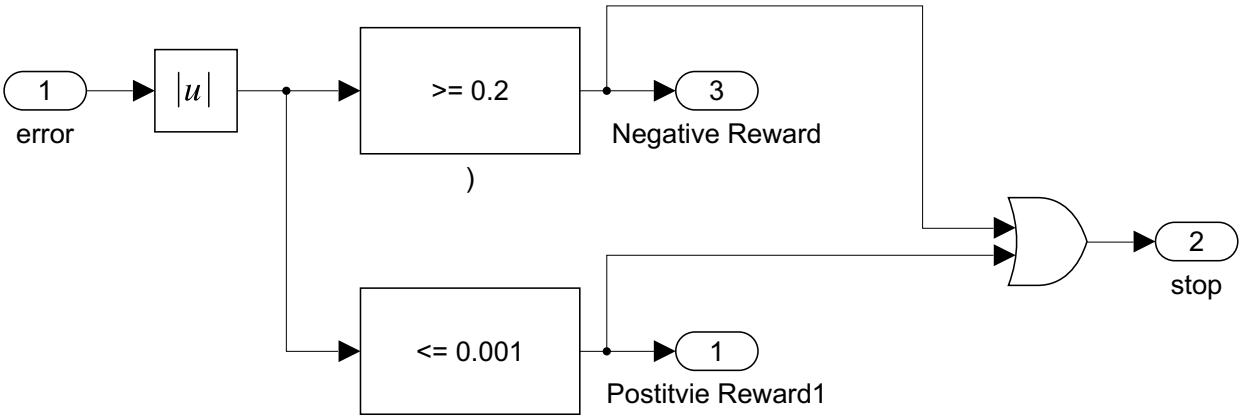


Figure 3.23: Limits Training 1

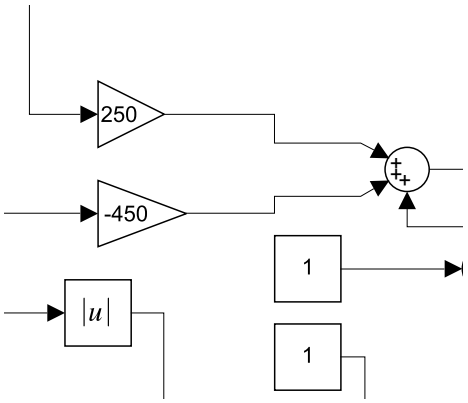


Figure 3.24: Reward Values, Training 1

3.8.3 Agent Tuning Part 2

Experience from agent tuning part 1 yields that the error value often equals 4 during the episode. This result in the termination off the training. Trying to avoid this problem the the author have adjusted the total number and off steps for each episode to 300. And in this training session the high limit value is increased to 0.1.figure 3.27 The reason is to try to make it easier for the agent to reach the target position.
Figure 3.25 Show the first 100 episodes for training 2.

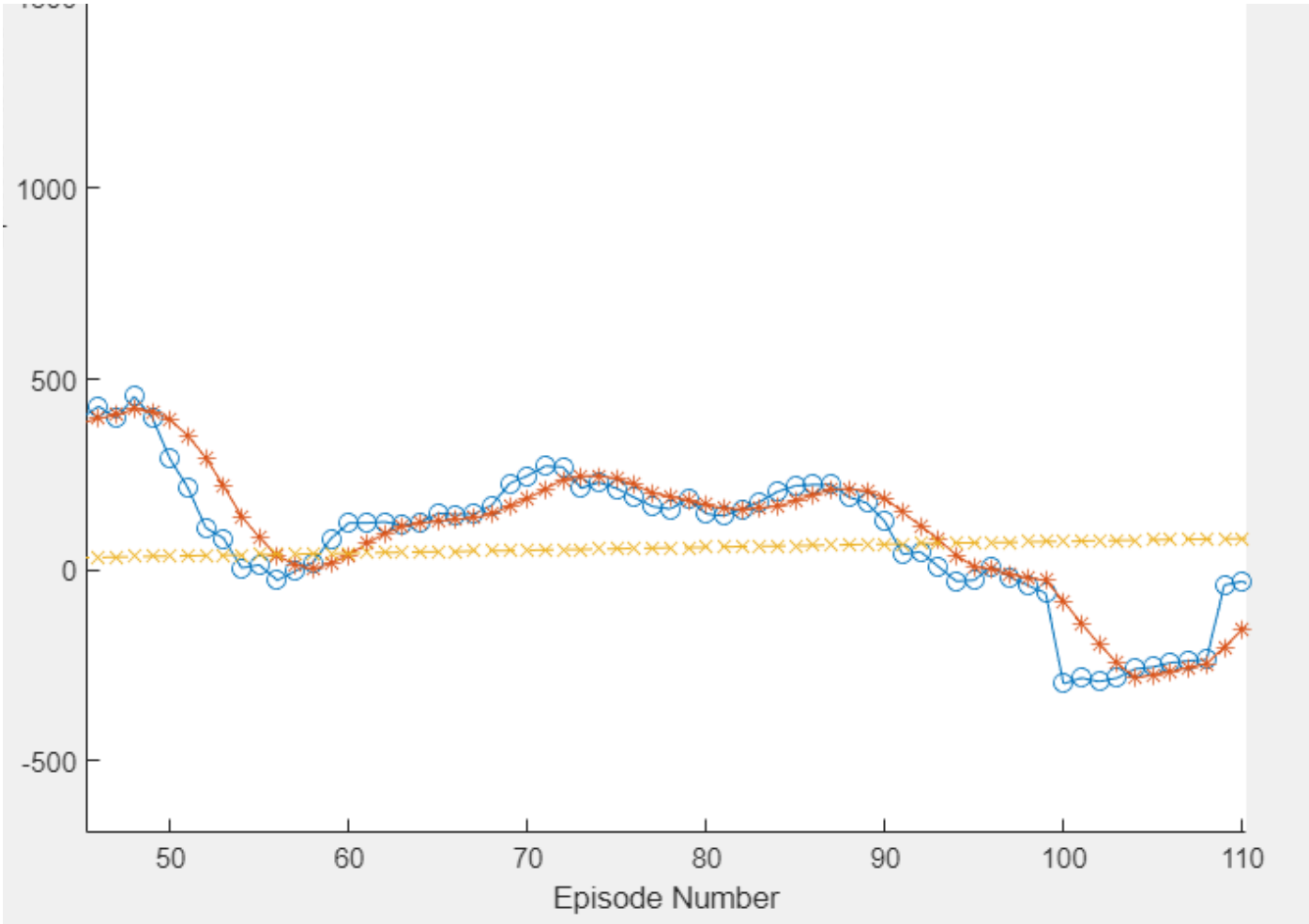


Figure 3.25: First 100 episodes

In the first 100 episodes the training gives some good results. But after the first 100 episodes there are almost none fluctuating in the total reward off each episode .

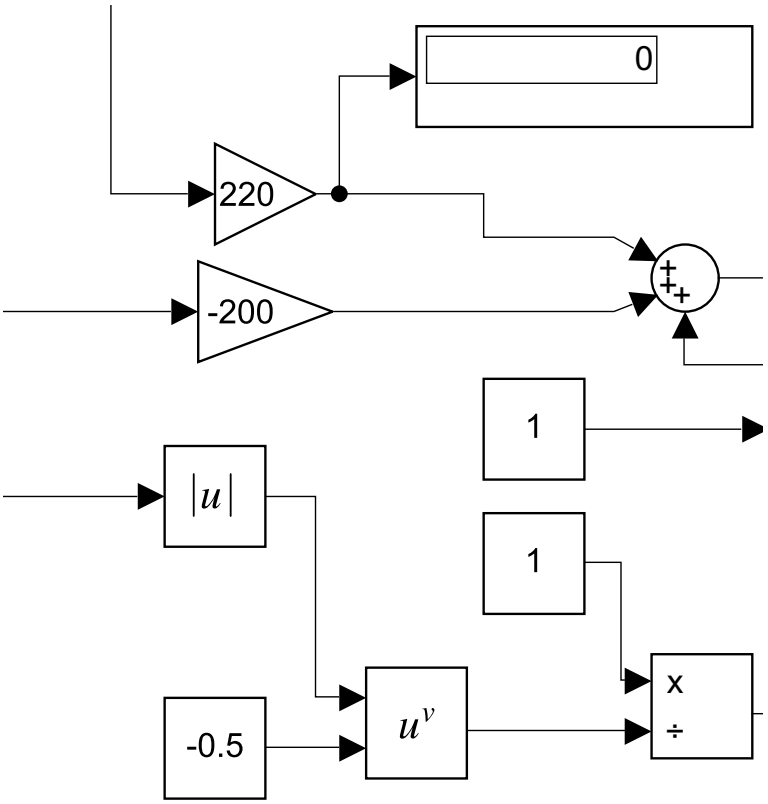


Figure 3.26: Reward values Training 2

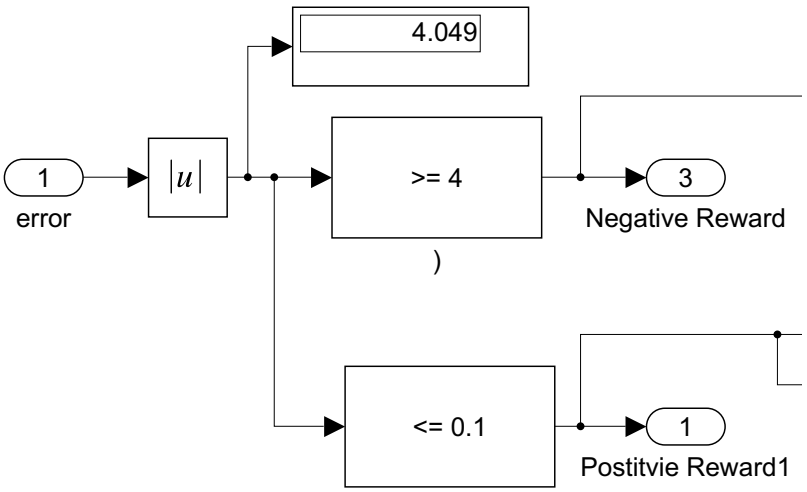


Figure 3.27: Limits Values for Training 2

3.8.4 Agent Tuning Part3

The values for this training session is:

lower limit: 0.001
upper limit: 3
Positive reward: 300
Negative reward: -200
Gain: 6

The result off this training is better. Now there are some more fluctuating in the total reward for each episode figure 3.28. The reason are the increased value off the gain.

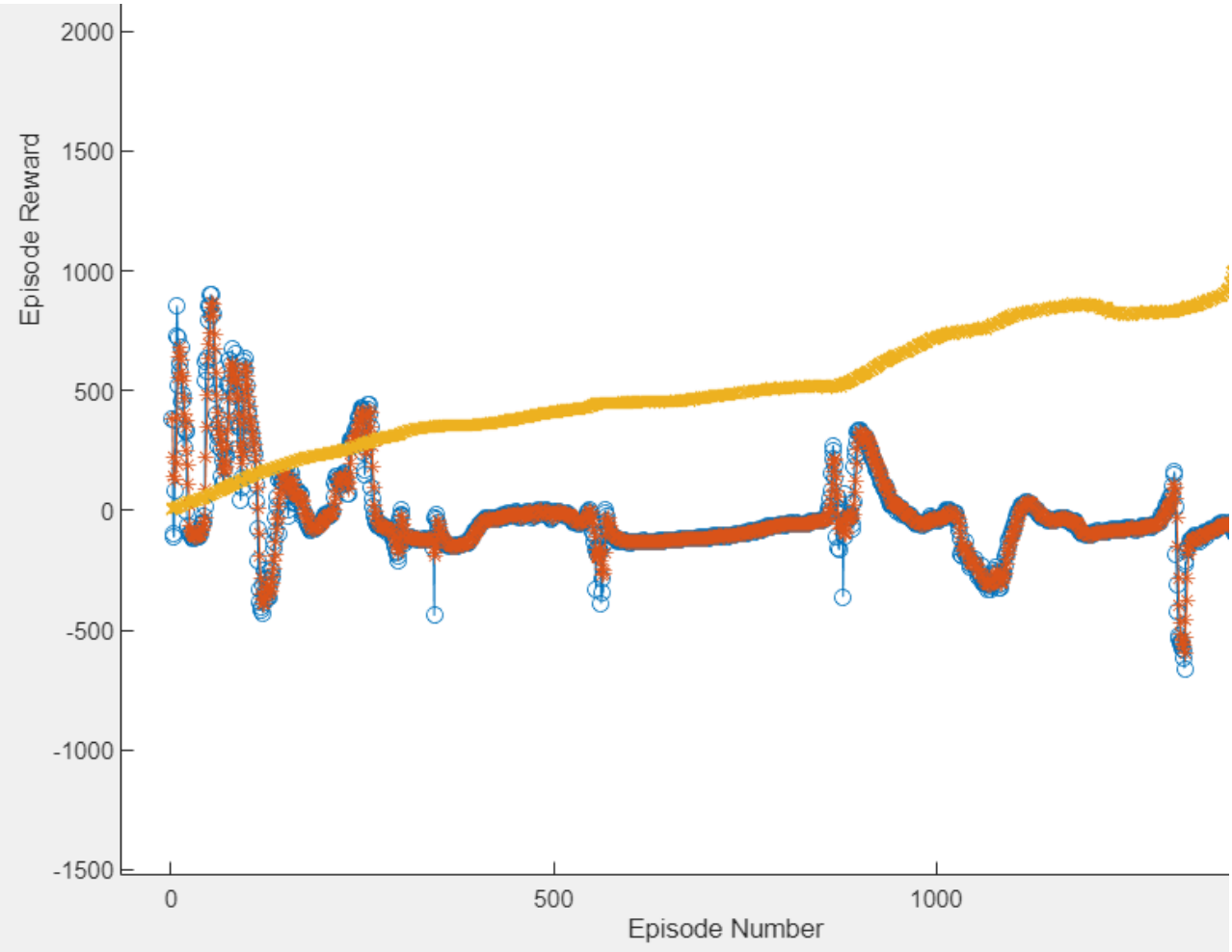


Figure 3.28: Result Training 3

3.8.5 Agent Tuning Part 4

Further research result in that for the previous training sessions the RL-agent is not able to reach the tolerance off 0.001 that result in the extra positive reward off 300. The RL-agent takes over the control off joint torque at waypoint 92. This is only 0.0162 meters above the target, see figure: 3.20 away from the target position. This does not give the agent enough time to adjust. So for this training session the conventional control is run to waypoint 75. At waypoint 75 the end-effector is 0.0505 meters (50mm) above the target position. This will maybe result in greater reward and that the agent is able to control the end-effector good enough to get the extra positive reward.

Figure 3.29 shows the joint configuration at waypoint 75. This values are put in as state target in the simscape model. Figure 3.30 shows the robot at the position that the RL-Agent start controlling the joint torques.

This training parameters did not yields in better results.

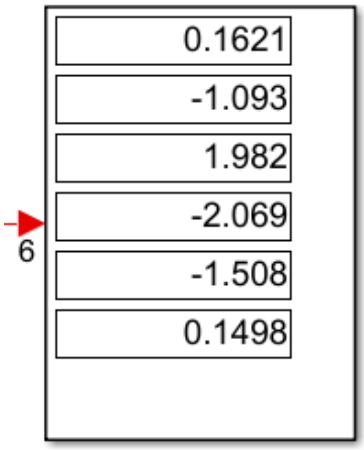


Figure 3.29: Configuration at waypoint 75

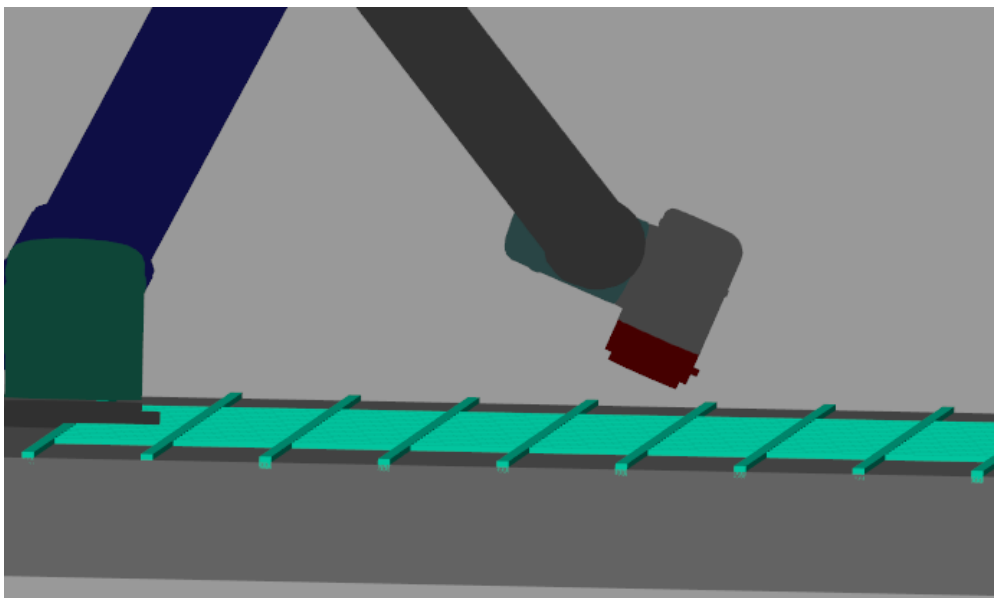


Figure 3.30: Robot position at waypoint 75

3.8.6 Agent Tuning Part 5

Trying to increase Agent ability to learn, the author choice to disable the agent ability to actuate joint 5 and 6 figure: 3.13. The author believe that this will give the agent less possibility to do actuation that only result in negative result. A other choice that is made is to run the RL-training for the waypoint 60 to further increase the time that the error value are decreasing and the reward value is increasing.

The values for this training session is:
Error value: Only the last element off the error matrix
lower limit: 0.008
upper limit: 0.3
Positive reward: 600
Negative reward: -500
Gain: 3
Waypoint start 60

The episode reward for 5000 episode is showed in figure 3.32 The reward for the training episodes before episode 1500 and after 2200 have flat trending (not increasing or decreasing) Around episode 1620 there are a sudden increase in the reward value. This is not a result of the agent reaching the lower limit of 0.008 meters.

The saved agent 1641 is loaded, and the simulation program is loaded. From figure 3.31 The error value only decrease to 0.024 (24mm) meters at 0.97 seconds. Observing the robot movement in the mechanic explorer, the robot end-effector is moving downwards the first 0.3 second. And when the joint torque compensate for the force from gravity, the end-effector moves towards the target. At 0.97 the end effector goes past the target position, and the distance to the target increases.3.31

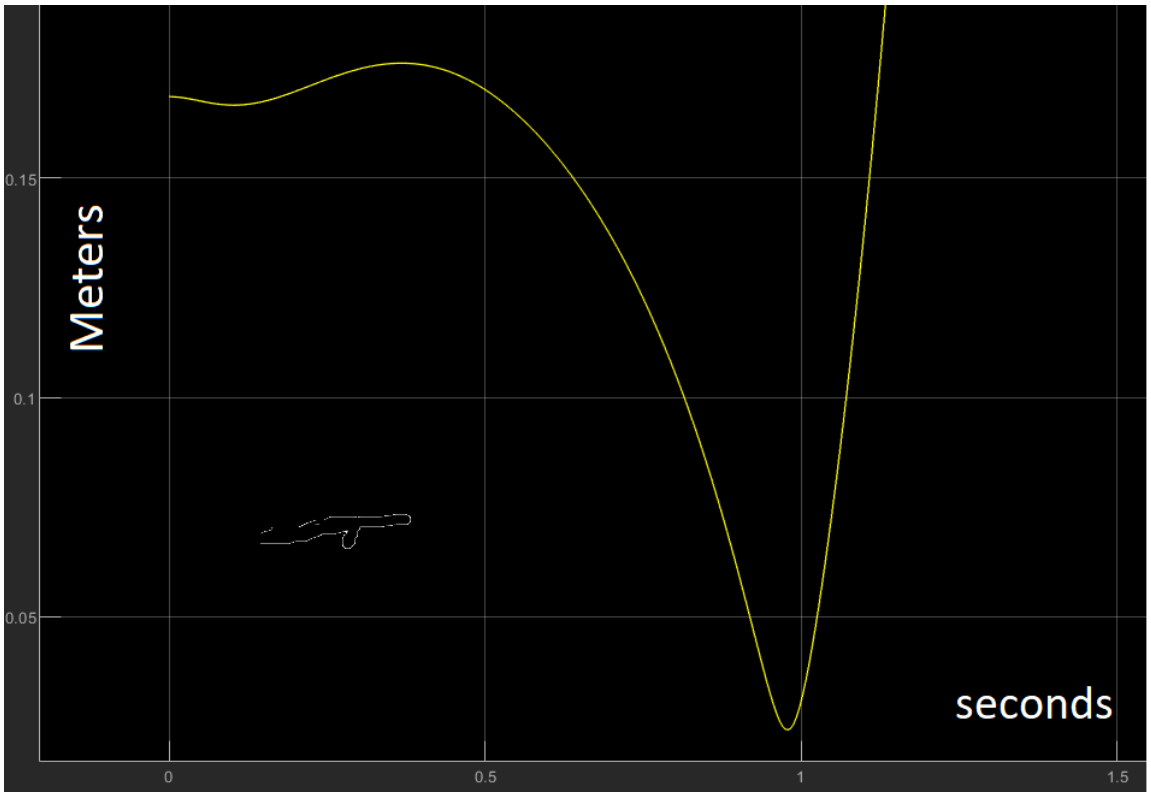


Figure 3.31: Episodes 1641 Minimum Error Value

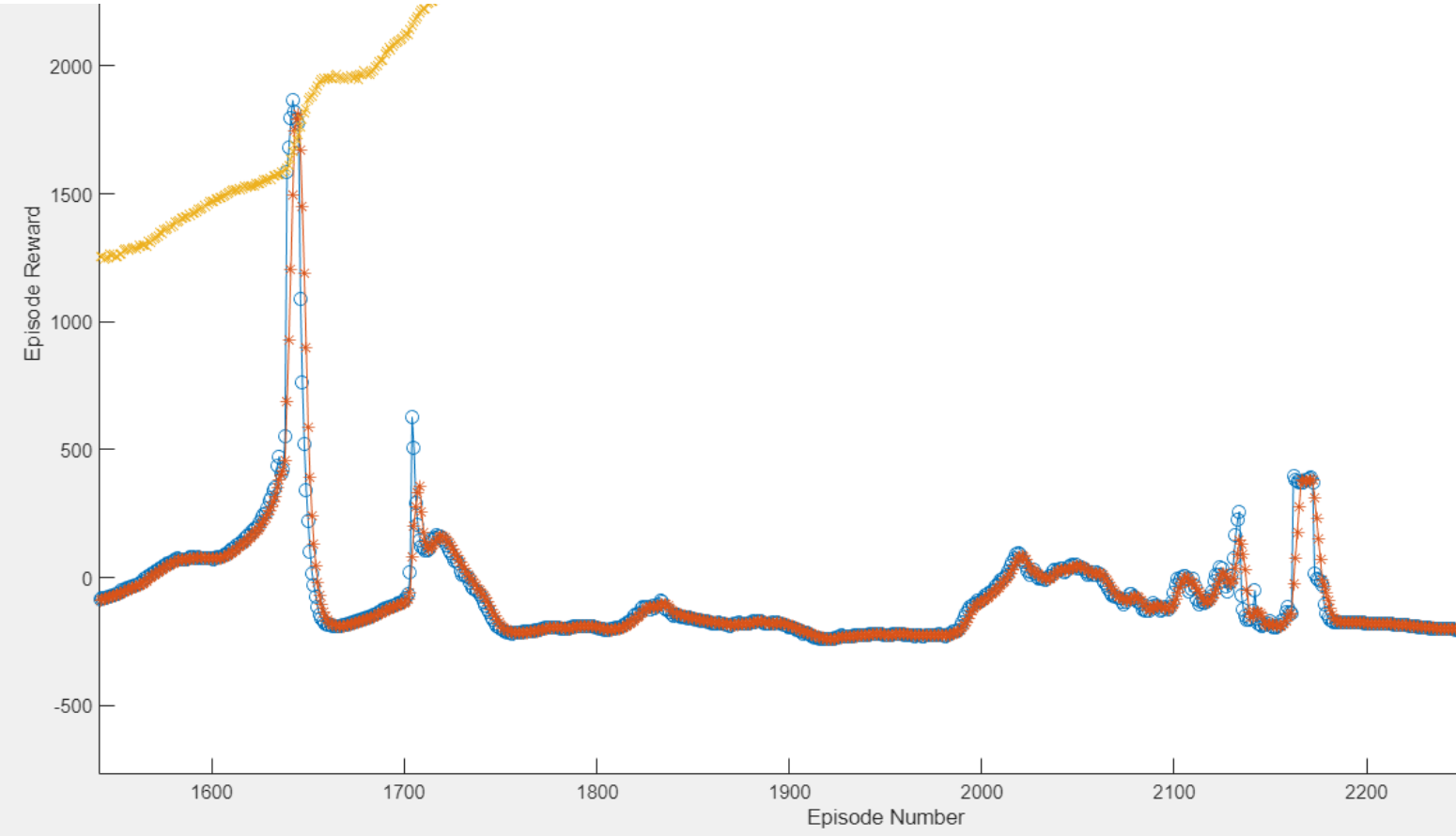


Figure 3.32: Episodes Reward For Training 5

3.8.7 Agent Tuning Part 6

In this session the lower limit is to the value close to the min value from training 5. Figure 3.31 This is a attempt to increase the reward value thus, increase the end result. Reaching the lower limit at 0018 will result in a sudden increase in the episode reward. This is not present in figure 3.33

The values for this training session is:
Error value: Only the last element off the error matrix 3.15
lower limit: 0.018
upper limit: 0.3
Positive reward: 600
Negative reward: -700
Gain: 5
Waypoint start 60
Total number of episodes 5000
Actuation of all six joints

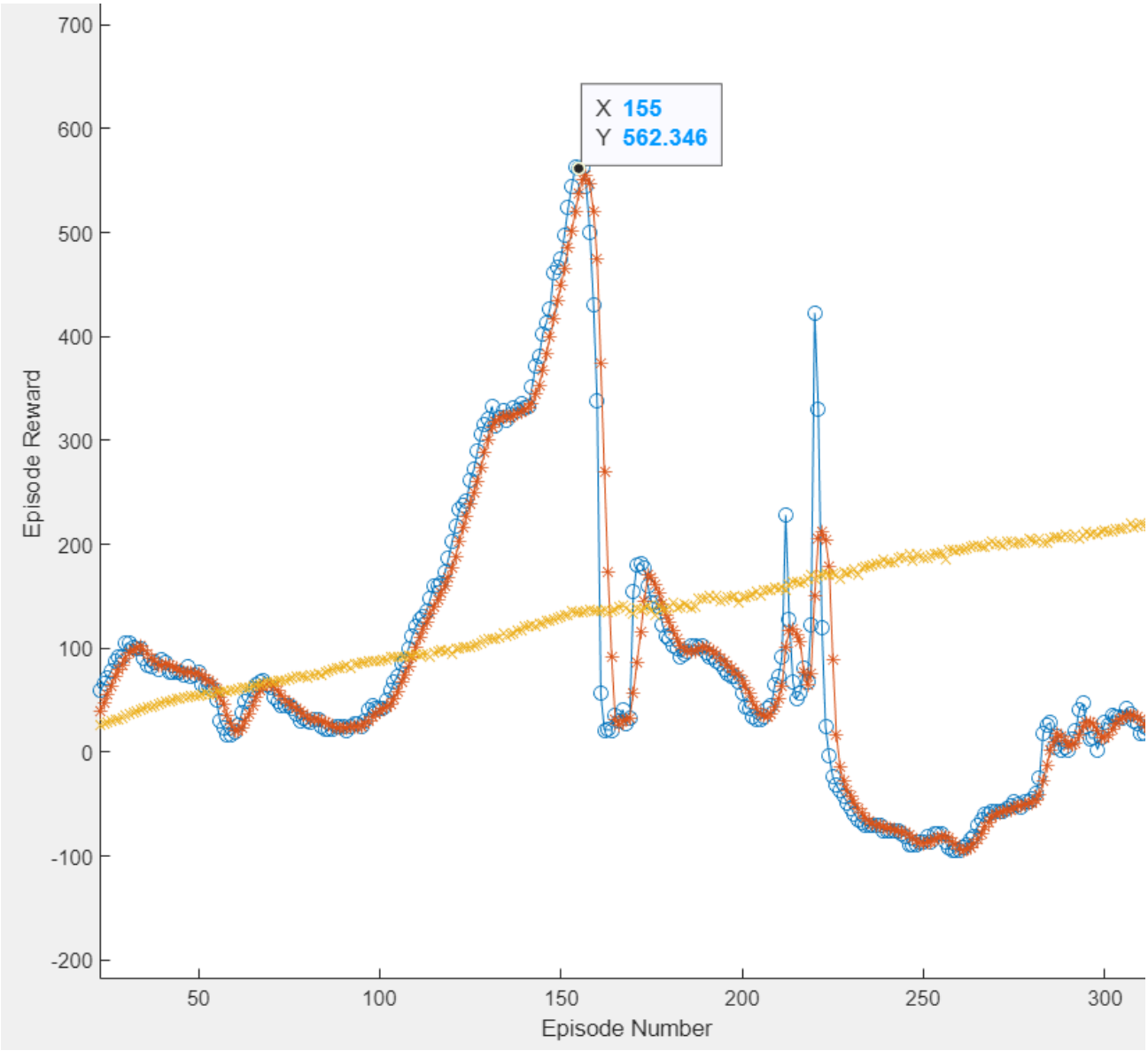


Figure 3.33: Episodes Reward For Training 6

3.8.8 Agent Tuning Part 7

In this part a extra limit value is added. The thought is to give the agent a reward when is it close to a limit, without the episode being terminated. See figure 3.34 The lower reward at 0.03 is added, this yields that the end-effector is 30mm from the target position when this limit is reach. If this limit is reach a extra reward value 600 is added to the total episode reward.Figure 3.35 This is a attempt to get a better end result of the learning session.

Figure 3.36 shows the learning episode manager for this session. There are some more variation in this session, there are episodes with high reward value at the first 400 episodes. (similar to the previously training session) But in this session there are also some episodes with high reward around episode 8500.

The values for this training session is:
Error value: Only the last element off the error matrix 3.15
Extra lower limit: 0.03
Termination Lower limit: 0.01 upper limit: 0.3
Extra reward: 600
Termination reward: 7000
Negative reward: -700
Gain: 5
Waypoint start 60
Total number of episodes 1000
Actuation of all six joints

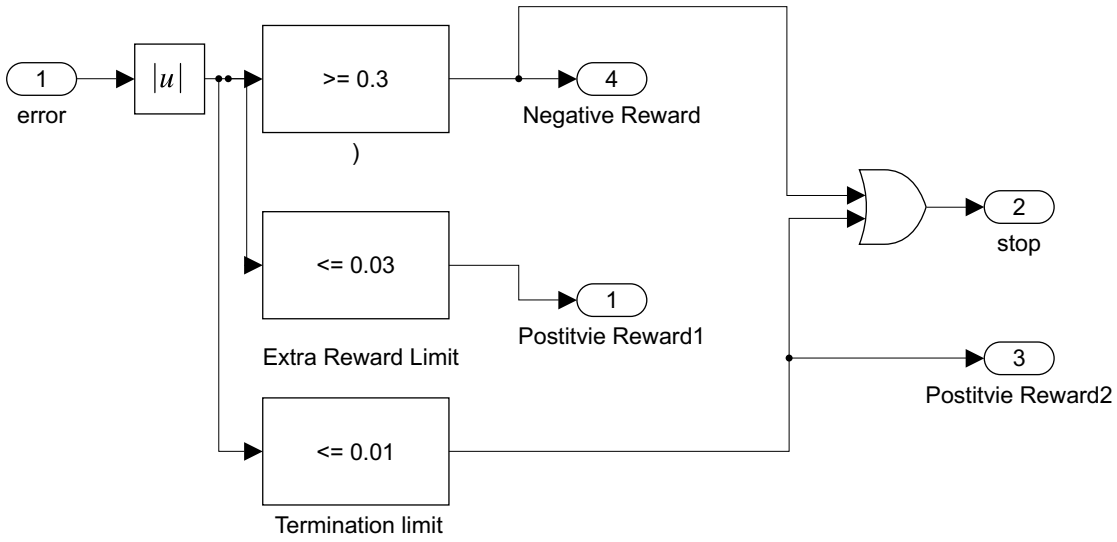


Figure 3.34: Reward Block with Extra Reward limit

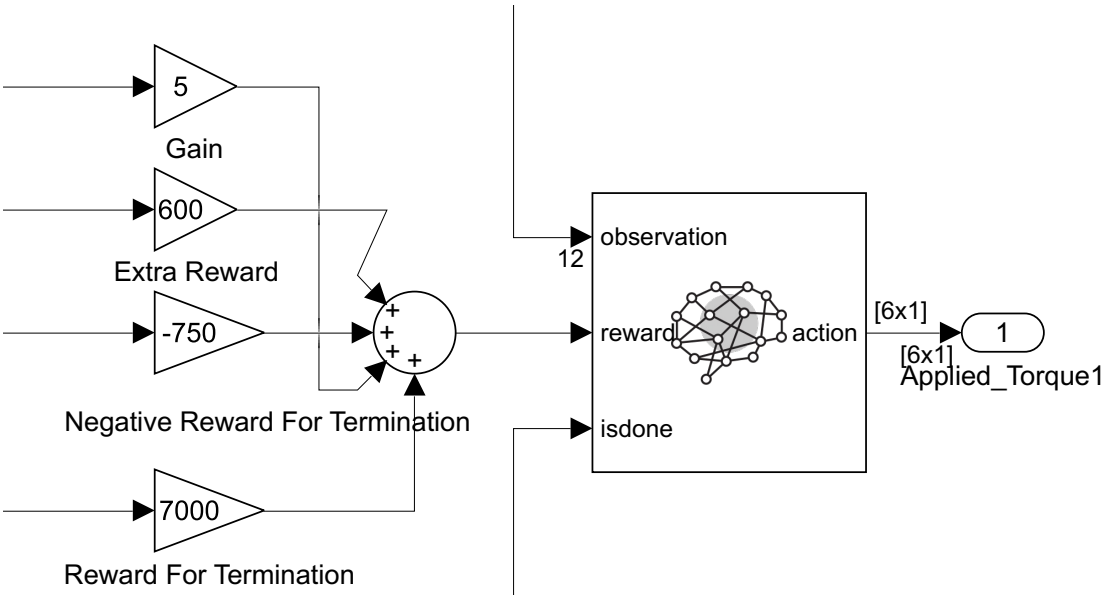


Figure 3.35: Values At The Different Limits

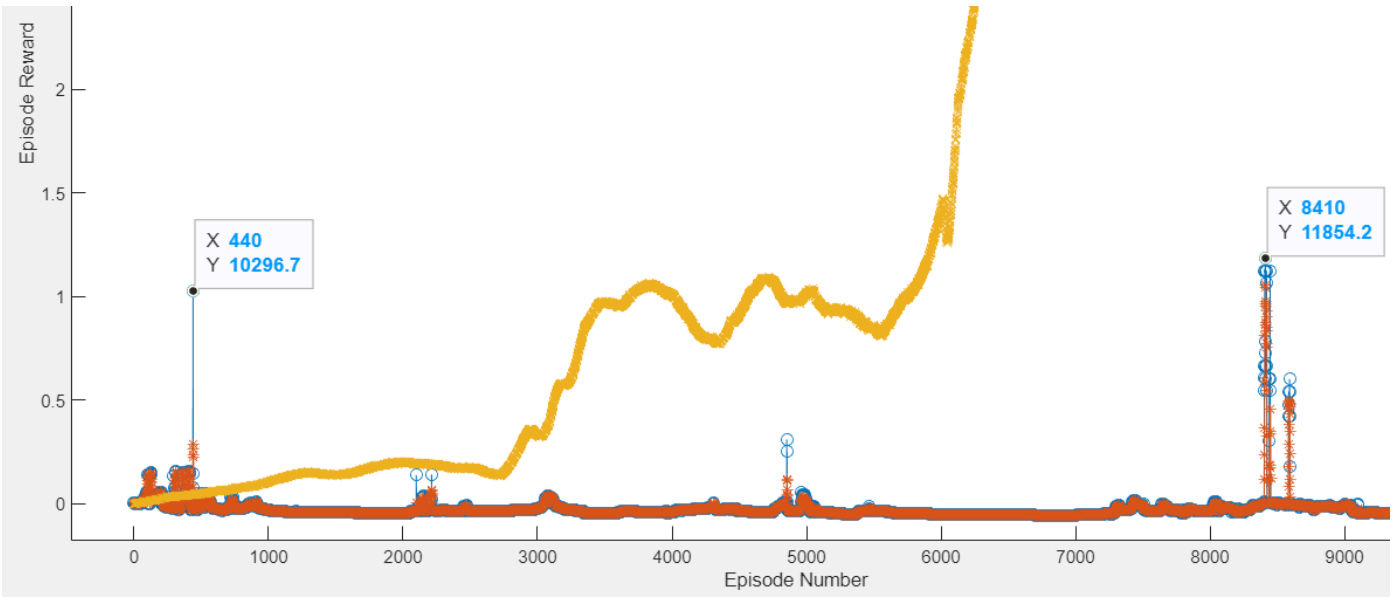


Figure 3.36: Learning From Episodes Training7

3.9 Combination off Conventional Control And Reinforcement Control

When implementing a combination conventional control and reinforcement control, some changes need to be done to the waypoint selector. Earlier version figure 3.2. The waypoint selector for the combination of reinforcement and conventional control showed in figure 3.37 Firs the simulation is runed until waypoint 60, where the agent is trained from. A new error matrix is constructed, error2. This error matrix is calculated considering the last waypoint (number100) that is the target position. This is same error matrix that is used during training of the RL-agent section.

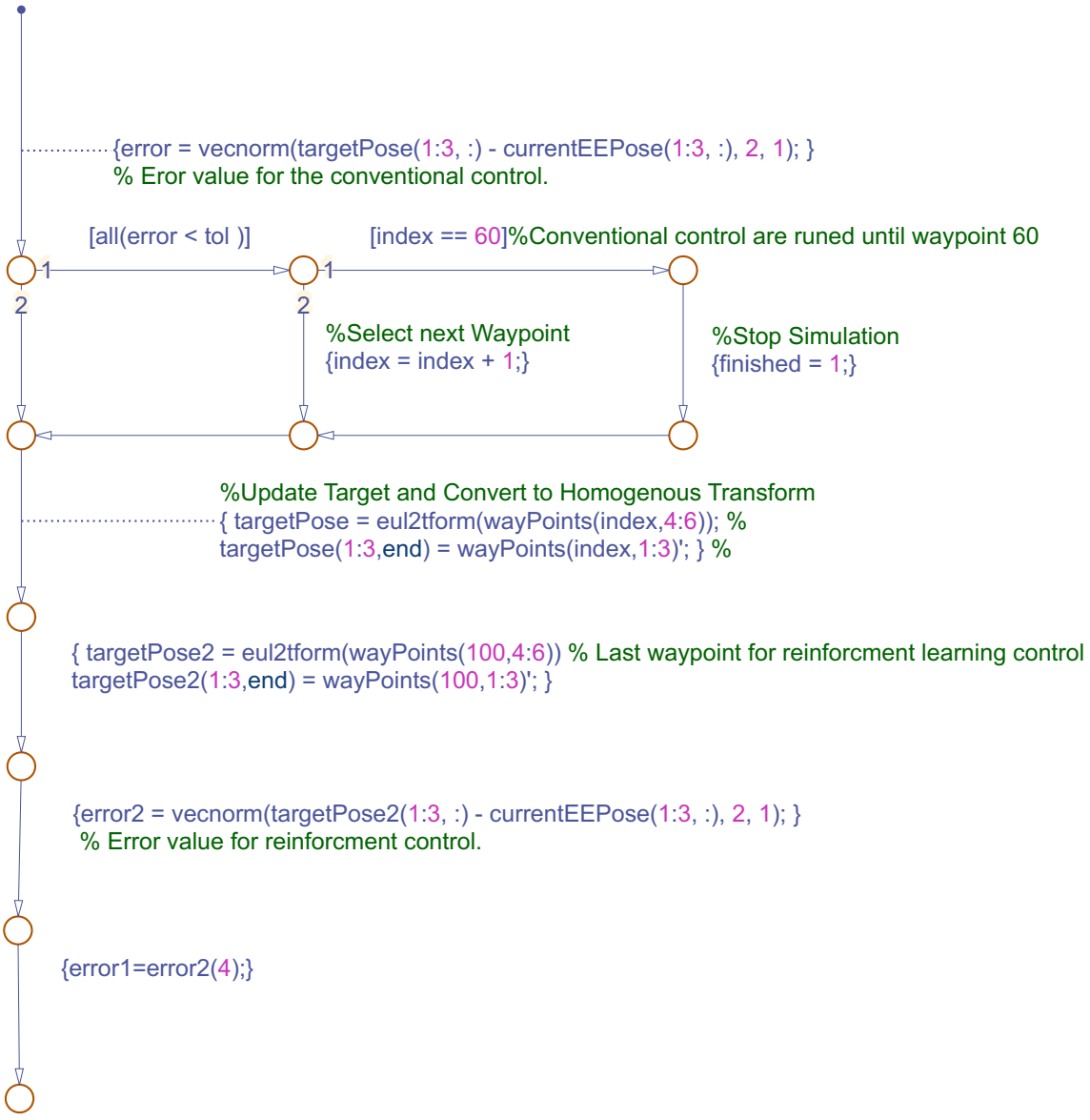


Figure 3.37: Waypoint selector for the combination of Reinforcment Control And Conventional

The workflow when implementing both conventional and reinforcement control is shown in 3.38

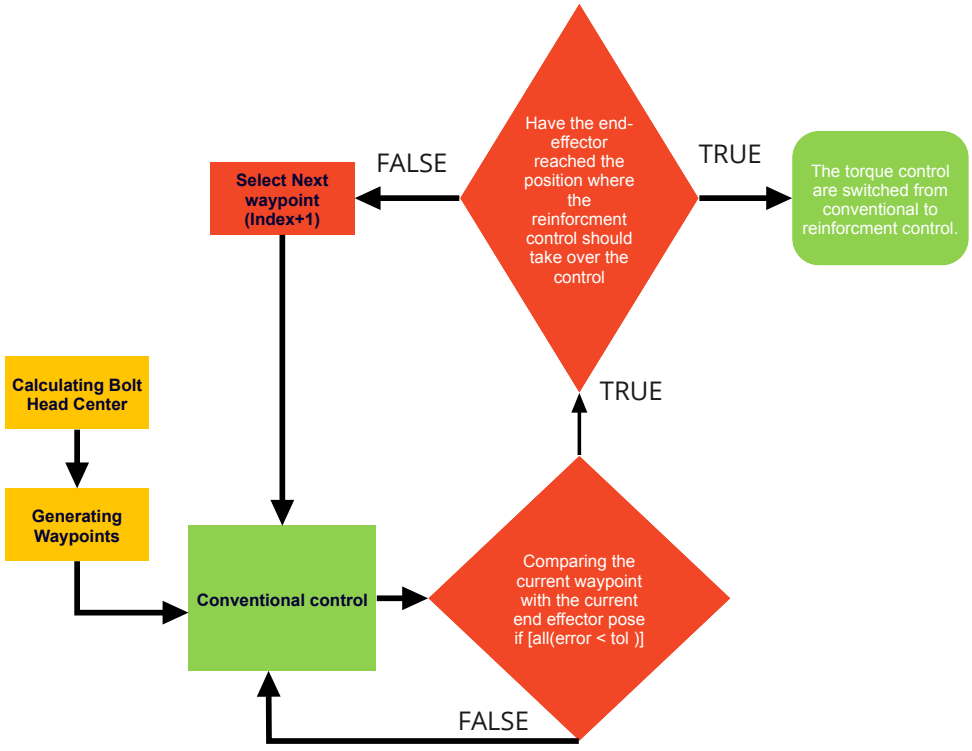


Figure 3.38: Control flowchart for both conventional control and Reinforcement control

Chapter 4

Results

4.1 Runing Script For Location Off Bolt Head, Conventional Control

The robot can visualized in simulink and simscape. In figure 4.1 the robot is visualized using the matlab script A. The visualisation part is the last section in the appendix A The robot is showed with the bolt location visible (white dots). The position off the bolt head 1 is 0.566, 0.1909 and 0.(x,y,z) The robot is placed at origo position.

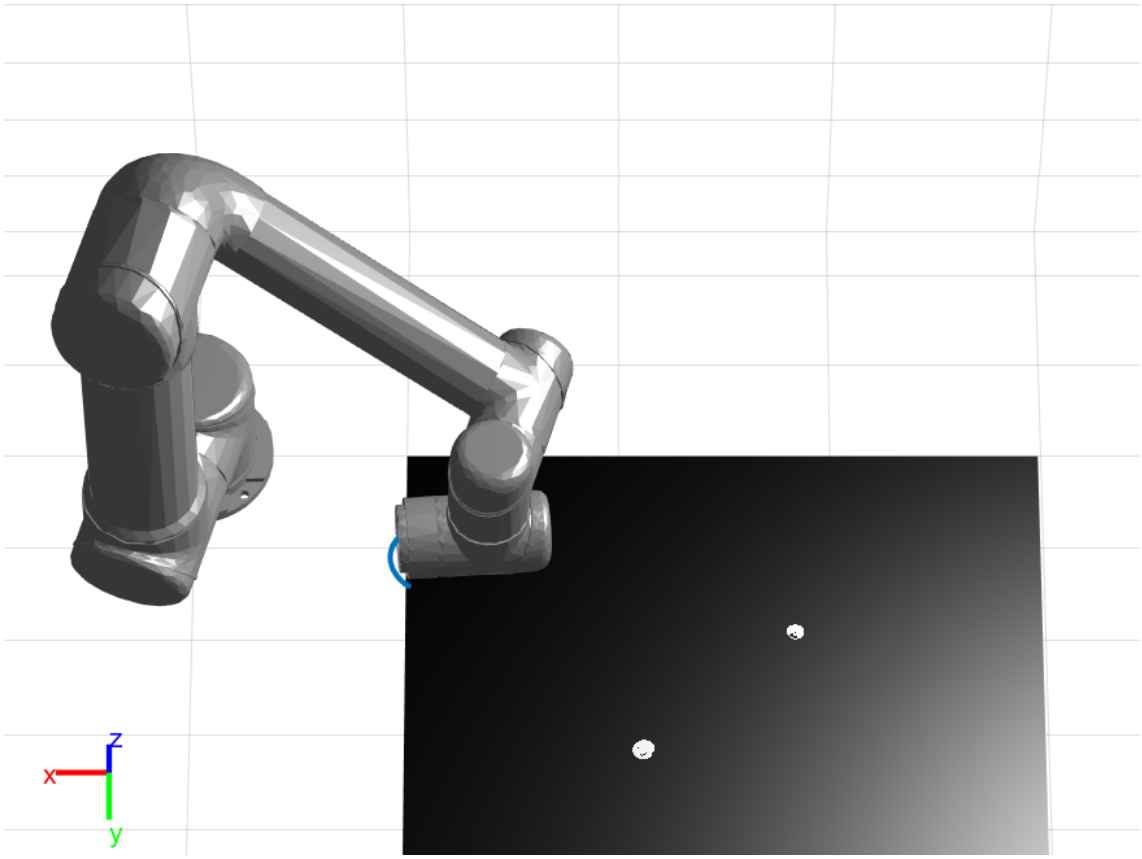


Figure 4.1: Robot in starting position

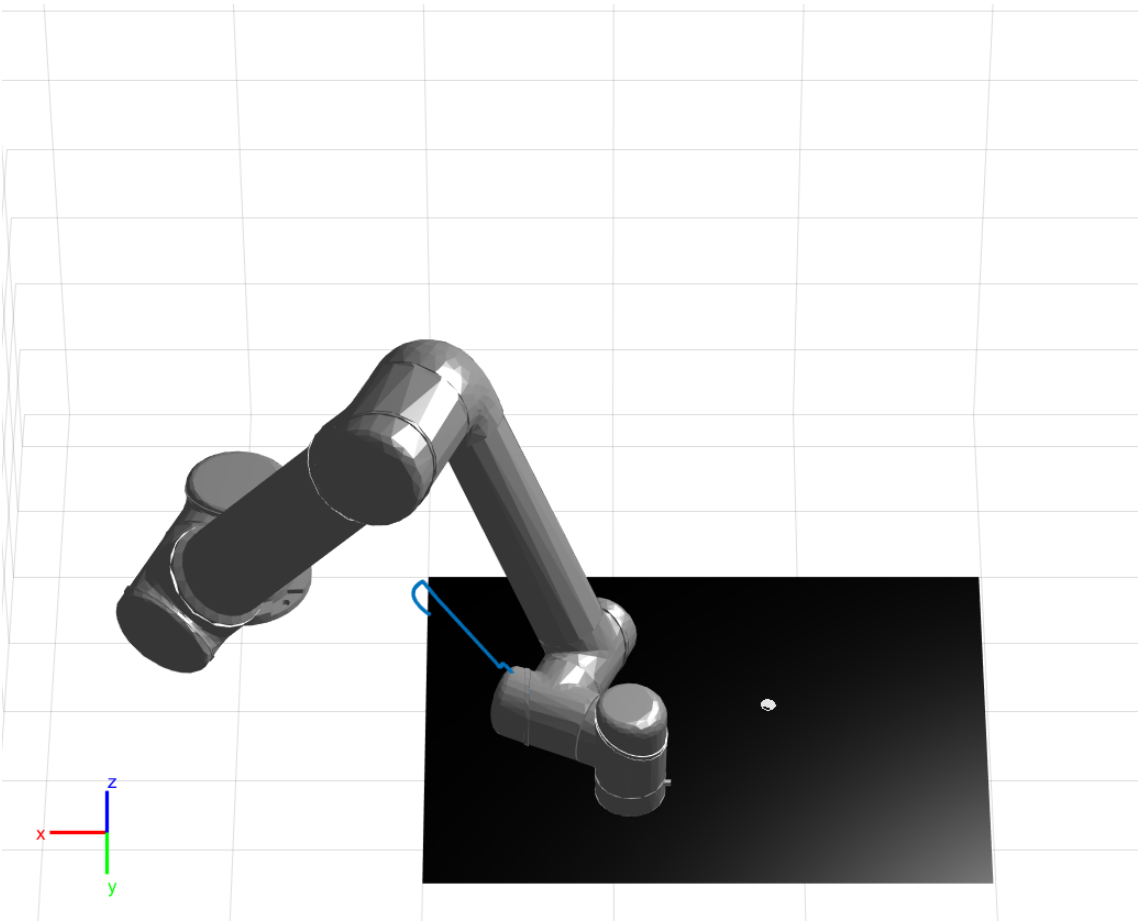


Figure 4.2: Robot at target location

4.2 Results From Conventional Control

From section 4.1 the location off the bolt head is at the cartesian coordinates 0.566, 0.1909 and 0. Running the script and generating the 100 waypoints towards the bolt head. The tolerance inside the waypoints selector figure: 3.2 is set to 0.001 that means that all the four elements in equation 4.1 is less than 0.001 when the waypoint selector selects the next waypoint. 0.001 correspond to a vector from the end-effector to the bolt head less than 0.001 meters (1mm). If the tolerance is set to less than 0.001 the simulation runs slower and the conventional control it is not able to control the joint torques good enough to get under the given tolerance value. Figure 4.3 shows the end results off the current end effector pose and the wanted targetpostion (the last waypoint). The last columns in each table represents the the cartesian coordinates.

$$error = \begin{bmatrix} E_1 & E_2 & E_3 & E_{x,y,z} \end{bmatrix} \tag{4.1}$$

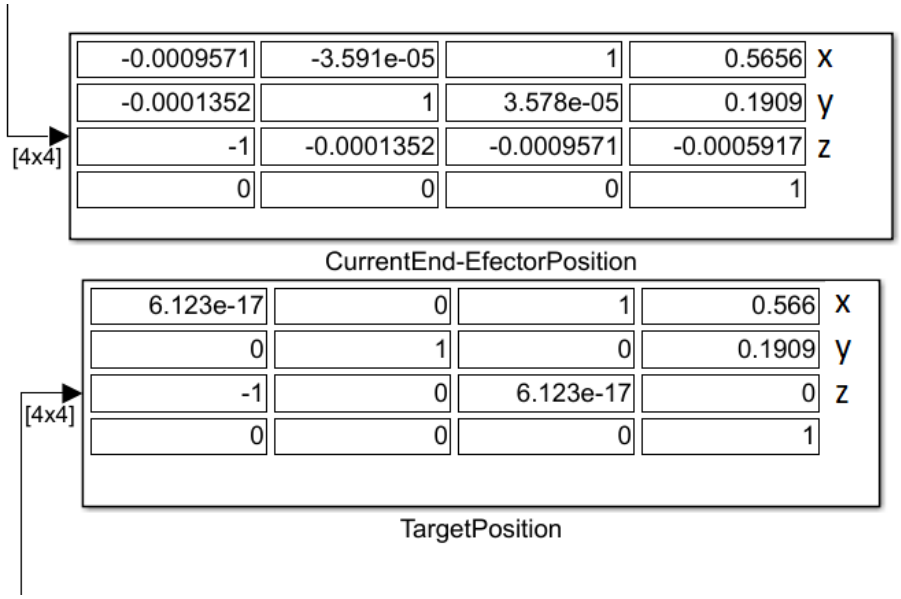


Figure 4.3: Result from simulation with conventional control

4.3 Results From Combination off Conventional Control And Reinforcement Control

Figure 3.36 Shows training results for training session 7. Number 8410 is chosen for further use. The agent number 8410 is uploaded in B and the simulation is done. The result of the agent training is not satisfying figure 4.5 shows the robot at target position after running simulation. In the appendix F the simulink files matlab scripts, and a video are included. The robot moves in a unnatural way towards the targetposition. This is because the orientation part of the error matrix, not is taken into account. The author have tried other agent form the last training session, but this are not satisfying either. But the agent do manage to get reduce the error to under 0.02 meters see figure 4.4. The result from agent tuning part 5 is better, but here the to last joints disabled. And this can not easily be combined with the conventional control. This is because of mismatch in matrix dimensions between the conventional control part and the reinforcement control.

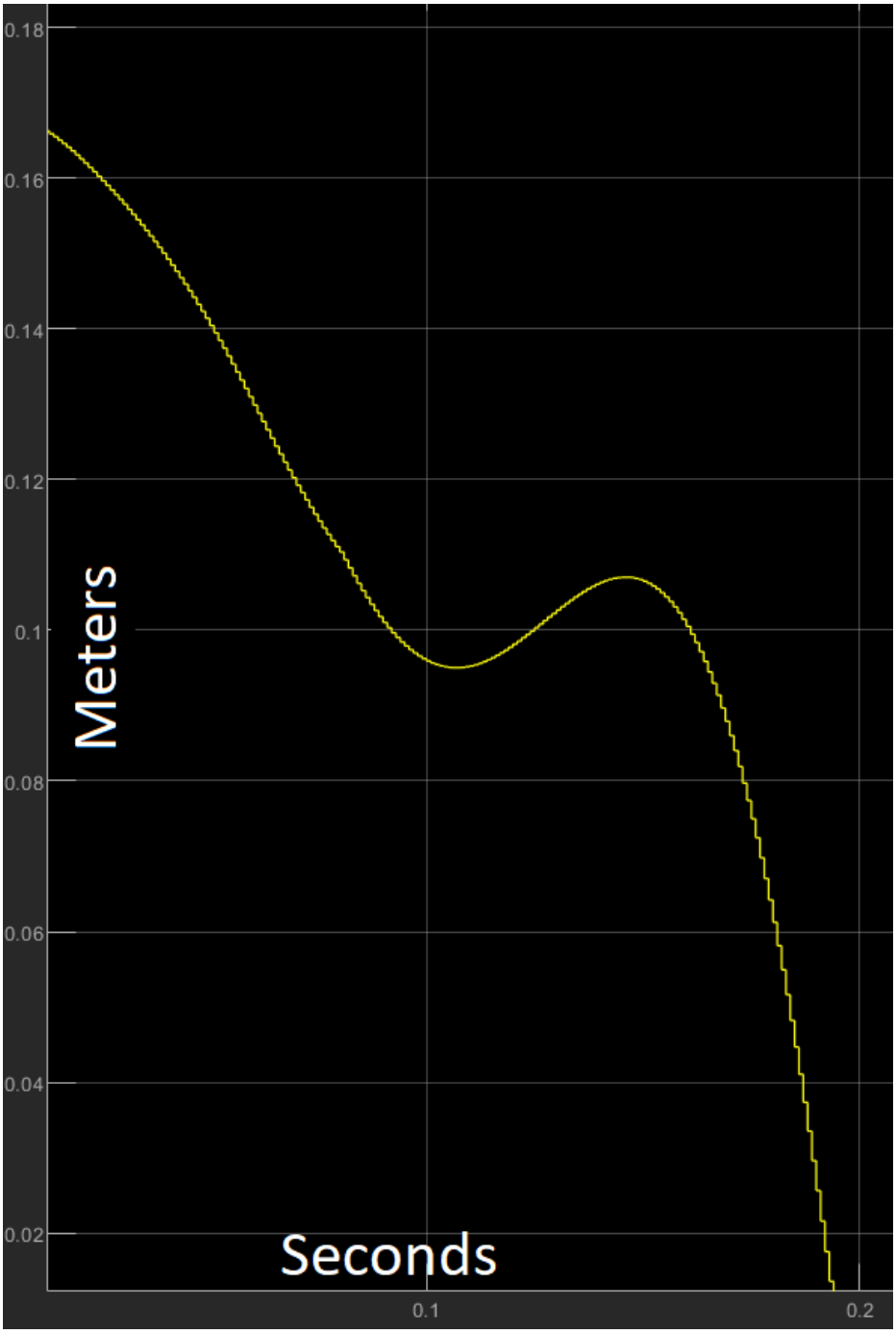


Figure 4.4: Error Value Distance to Target Position

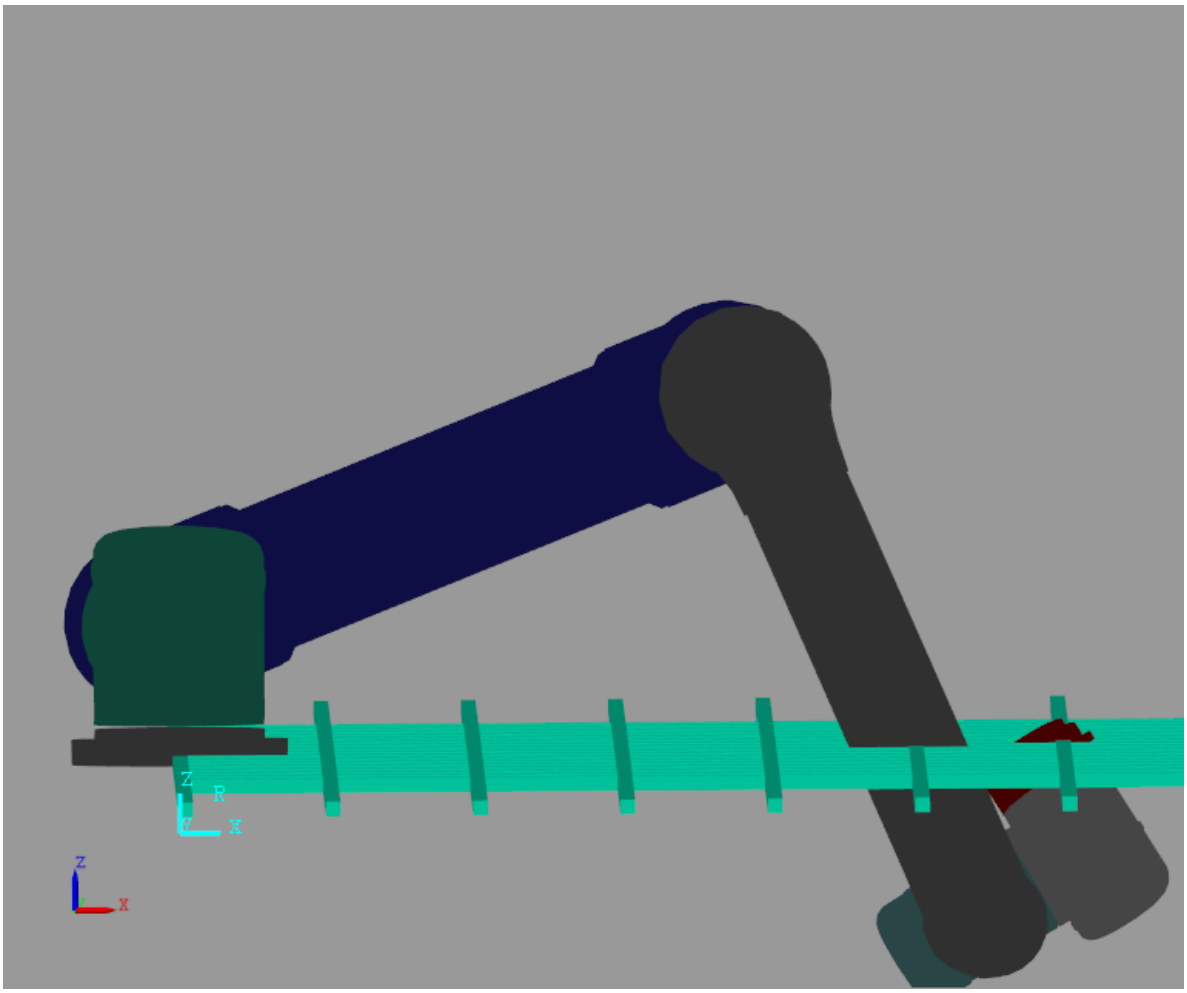


Figure 4.5: Mechanic explorer View Of UR-5 robot, end effector at target position after a combination of conventional and reinforcement control

Chapter 5

Discussions

5.1 Combination of Conventional Control And Reinforcement Control

In the end result the RL-Agent moves the robot joints in a unnatural way.This can be viewed in the video in the appendix F. This can be improved if the last to joint are disabled. This is done in The Agent Tuning Part 5. So in further work this can be improved. The accuracy for only only the conventional control is 1 mm. For the combination of conventional control and reinforcement control the accuracy is 13mm, and the end-effector have wrong orientation. For further work including the orientation elements of the error in the reward calculation should be included.

$$error = \begin{bmatrix} E_1 & E_2 & E_3 & E_{x,y,z} \end{bmatrix} \tag{5.1}$$

Regarding the bolt head location, further improvement could be to add a location of bolt head in a 3 dimensions. The solution in this paper is only in 2 dimensions.

Chapter 6

Conclusions

This paper show how to implement a combination of conventional and reinforcement learning based control of UR5 industrial robot

The combination of conventional control and reinforcement control are implemented. The combination of the to control parts works. But there are some issues regarding the control accuracy and orientation from the reinforcement control.

The result for the location of bolt head is satisfactory. The exact position for the bolt head is located, and is merged with the waypoint generation. The cartesian coordinates are put in as the last waypoint, thus the target for the end-effector.

In this study, reinforcement learning only works in the situation it is trained to perform. In other words, it does not work that well to control a UR-5 robot. The accuracy is much better with only conventional robotic control. The author believe that it could work better, if it were given other configurations/parameters.

The UR-5 robot have 6 degrees of freedom, this result in complex control task. There are almost infinity number of possible configuration sequences, for a given end-effector position. In further work it can be important to try to limit the amount of data given to the reinforcement learning algorithm. So this will still be an interesting subject for further studies. The author have tried to limit the number of joints that are actuated by the RL-Agent, but this is difficult to combine with the convectional control, but it yields is better result.

Further work

Many different parameter and configuration for the reinforcement learning part have not been implemented due to lack of time. Some are tried out, but some where hard to implement taken into account that it should work together with the conventional control. The author will point out some configuration that could improve the results from reinforcement control.

- In general reduce the amount of date given to the agent.
- Disable the last to joints, or include the orientation part off the error matrix in the reinforcement reward.
- Agent training with only the angular position θ, training in this paper include angular position θ and angular velocity $\dot{\theta}$.

Appendix A

Location Of Bolt, Waypoints Generation And Robot Visualizations

Based upon [5],[7],[9]

```
% Modified by  Fredrik Frigstad, based on the mathwork example listet above.

%% Loading URDF file,and adding datat to the simulink program
% Import the manipulator as a rigidBodyTree Object
robot = importrobot('ur5_robot.urdf');
%[robot,importInfo] = importrobot(simscape)

robot.DataFormat = 'column';
robot.Gravity = [0 0 -9.81]
% Define end-effector body name
eeName = 'ee_link';
%[H,dataFileName] = smimport(simscape123.slxc)
% Define the number of joints in the manipulator
numJoints = 6;
eeOrientation = [0, pi, 0];% The Euler angles for the desired end effector ...
    orientation at each point must
% also be defined.
clear wayPoints
%%
%  BOLT DETECTION

RGB = imread('final2.png');
imshow(RGB);
I = im2bw(RGB);
%imshow(I);
bw= imcomplement(I); % convert black and white
BW2 = bwareaopen(bw,300); %150000     470000%
% bw = imfill(BW2,'holes');
% imshow(bw)
[B,L] = bwboundaries(BW2,'noholes');
%imshow(label2rgb(L,@jet,[.5 .5 .5]))
hold on
for k = 1:length(B)
  boundary = B{k};
  plot(boundary(:,2),boundary(:,1),'w','LineWidth',2)

end
scale = 0.0015/2;% 0.00015 med bilde boltgreen  0.000075/2  0.00035
segment = B{1}*scale;    % translate from pixels to distance (m)

%segment2 = B{2}*scale;
```

```
% bw_filled = imfill(bw,'holes');
% boundaries = bwboundaries(bw_filled);
imageOrigin = [0.2,0,0.0];

    % Z-offset for moving between boundaries
  segment(1,3) = .0;
  segment2(1,3) = .0;
%       Translate to origin of image
    cord = imageOrigin + segment;
    cord2 = imageOrigin + segment2;
%       bolt origin x cord
%%
% BOLT LOCATION

% bolt 1 location in global kordinate system
   Bolt1xWidth = max(cord(:,1))-min((cord(:,1)));  % bolt 1 width = x ...
      maxValue - x minValue
   Bolt1xLocation=min(cord(:,1))+Bolt1xWidth/2;    % bolt1 x cordinates = ...
      start off left bolt + bolt width

   Bolt1yWidth = max(cord(:,2))-min((cord(:,2)));
   Bolt1yLocation=min(cord(:,2))+ Bolt1yWidth/2;
   Bolt1Center(1,1)=Bolt1xLocation;
   Bolt1Center(1,2)=Bolt1yLocation;
% bolt2 location in global kordinate system

   Bolt2xWidth = max(cord2(:,1))-min((cord2(:,1)));   % bolt 2width = x ...
      maxValue - x minValue
   Bolt2xLocation=min(cord2(:,1))+Bolt2xWidth/2;    % bolt2x cordinates = ...
      start off left bolt + bolt width

   Bolt2yWidth = max(cord2(:,2))-min((cord2(:,2)));
   Bolt2yLocation=min(cord2(:,2))+ Bolt2yWidth/2;

   Bolt2Center(1,1)=Bolt2xLocation;
   Bolt2Center(1,2)=Bolt2yLocation;

%%
 % WAYPOINT GENERATION

% Start just above image origin
%waypt0 = [imageOrigin + [0 0 .2],eeOrientation]
waypt0 = [0.2 0.2 0.2,eeOrientation]% where to start
%imageOrigin = [0.6 -0.3 0.05];
Bolt1Center(1,3) = .0;
Bolt2Center(1,3) = .0;   %  add column

waypt1 = [Bolt1Center,0,pi/2,0];%  Target location
%waypt1 = [imageOrigin+Bolt2Center,eeOrientation];%  Target location
% Interpolate each element for smooth motion to the origin of the image
for i = 1:6
    interp = linspace(waypt0(i),waypt1(i),100);
    %eulerAngles = repmat(eeOrientation,size(segment,1),1);
    wayPoints(:,i) = interp';
end

q0 = zeros(numJoints,1);

% Define a sampling rate for the simulation.
%Ts = .001;
```

```
weights = [1 1 1 1 1 1];
% Transform the first waypoint to a Homogenous Transform Matrix for ...
    initialization
%
initTargetPose = eul2tform(wayPoints(1,4:6));
%
initTargetPose(1:3,end) = wayPoints(1,1:3)';
%
% Solve for q0 such that the manipulator begins at the first waypoint
 ik = inverseKinematics('RigidBodyTree',robot);
 [q0,solInfo] = ik(eeName,initTargetPose,weights,q0);

%
% targetPose = eul2tform(wayPoints(1,4:6))
%  wayPoints(1,4:6)
% targetPose(1:3,:)
% wayPoints(1,1:3)'
%  error = vecnorm(targetPose(1:3,:) - currentEEPose(1:3,:), 2, 1)

% Remove unnecessary meshes for faster visualization

%sim('robotsim.slx');
%

%%      VISUALIZATIONS OF ROBOT

% show(robot);
% xlim([-1.00 1.00])
% ylim([-1.00 1.00]);
% zlim([-1.02 0.98]);
% view([128.88 10.45]);
%clearMeshes(robot);

% Data for mapping image
% [m,n] = size(BW2);
%
% [X,Y] = meshgrid(0:m,0:n);
% X = imageOrigin(1) + X*scale;
% Y = imageOrigin(2) + Y*scale;
% Z = zeros(size(X));
% Z = Z + imageOrigin(3);
% % Close all open figures
% close all
% % Initialize a new figure window
% figure;
% set(gcf,'Visible','on');
%
% % Plot the initial robot position
% show(robot, jointData1(1,:)');
% hold on
% % Initialize end effector plot position
% p = plot3(0,0,0,'.');
% warp(X,Y,Z,BW2');
%
% % Change view angle and axis
% view(0,45)
% axis([-0.25 1 -0.25 0.6 0 0.75])
%
% % Iterate through the outputs at 10-sample intervals to visualize the ...
    results
% for j = 1:10:length(jointData1)
```

```
%     % Display manipulator model
%     show(robot,jointData1(j,:)', 'Frames','off','PreservePlot', false);
%     %show(robot,config,'Collisions','on','Visuals','off');
%     % Get end effector position from homoegenous transform output
%     pos = poseData1(1:3,4,j);
%      %'Frames', 'off','Collisions','on' 'PreservePlot', false);
%     % Update end effector position for plot
%     p.XData = [p.XData pos(1)];
%     p.YData = [p.YData pos(2)];
%     p.ZData = [p.ZData pos(3)];
%     % Update figure
%     drawnow
% end
```

Appendix B

Generating DDPG Agent and Train Agent Matlab Script

Based upon: [6]

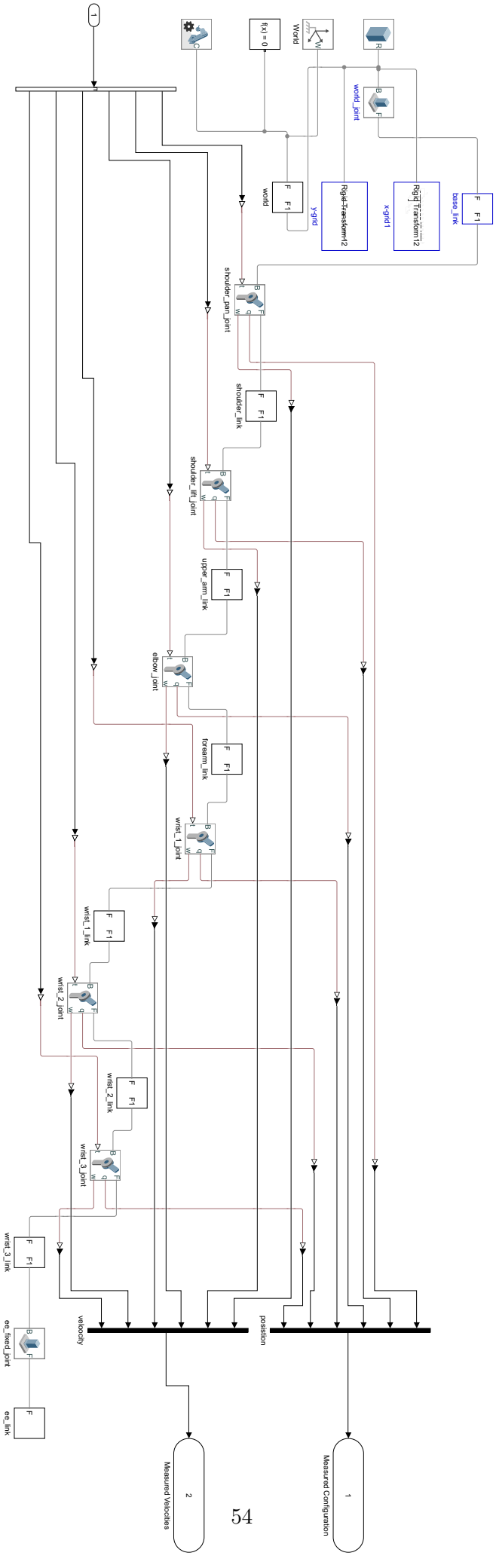
```
Some small modification is done by Fredrik Frigstad, the rest is the ...
    example listed above.

% error=[0:0.01:0.3]

%  reward=(1-(1./0.01).^-0.5)*3  %reward function validation

%  rewardtest=(1-(1/0.002)^-0.5)*8

%plot(error,reward)
%open_system('robotsim')
%open_system('robotsimTrain')
obsInfo = rlNumericSpec([12 1],...
      'LowerLimit',[-inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf ...
         -inf]',...
      'UpperLimit',[ inf inf inf inf inf inf inf inf inf inf inf inf]');
 obsInfo.Name = 'observations';
 obsInfo.Description = 'conf, vel'; % legge inn alle 12 verdiene
 numObservations = obsInfo.Dimension(1);

actInfo = rlNumericSpec([6 1]);
actInfo.Name = 'Torque';
numActions = actInfo.Dimension(1);

%env = rlSimulinkEnv('robotsim','robotsim/EnabledReinforcment/RL ...
    Agent',obsInfo,actInfo);

env = ...
    rlSimulinkEnv('robotsimRein25_05','robotsimRein25_05/EnabledReinforcment/RL ...
    Agent',obsInfo,actInfo);

%env.ResetFcn = @(in)localResetFcn(in);% The robot is put in place close to
%the target screw, so this is not needed.

Ts = 0.001;
Tf = 40;

rng(0)


% CREATE AGENT DDPG
```

```
statePath = [
    featureInputLayer(numObservations,'Normalization','none','Name','State')
    fullyConnectedLayer(50,'Name','CriticStateFC1')
    reluLayer('Name','CriticRelu1')
    fullyConnectedLayer(25,'Name','CriticStateFC2')];
actionPath = [
    featureInputLayer(numActions,'Normalization','none','Name','Action')
    fullyConnectedLayer(25,'Name','CriticActionFC1')];
commonPath = [
    additionLayer(2,'Name','add')
    reluLayer('Name','CriticCommonRelu')
    fullyConnectedLayer(1,'Name','CriticOutput')];

criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork,statePath);
criticNetwork = addLayers(criticNetwork,actionPath);
criticNetwork = addLayers(criticNetwork,commonPath);
criticNetwork = connectLayers(criticNetwork,'CriticStateFC2','add/in1');
criticNetwork = connectLayers(criticNetwork,'CriticActionFC1','add/in2');

criticOpts = rlRepresentationOptions('LearnRate',1e-03,'GradientThreshold',1);

critic = ...
   rlQValueRepresentation(criticNetwork,obsInfo,actInfo,'Observation',{'State'},'Action

%construct actor

actorNetwork = [
    featureInputLayer(numObservations,'Normalization','none','Name','State')
    fullyConnectedLayer(3, 'Name','actorFC')
    tanhLayer('Name','actorTanh')
    fullyConnectedLayer(numActions,'Name','Action')
    ];

actorOptions = ...
   rlRepresentationOptions('LearnRate',1e-04,'GradientThreshold',1);

actor = ...
   rlDeterministicActorRepresentation(actorNetwork,obsInfo,actInfo,'Observation',{'Stat


agentOpts = rlDDPGAgentOptions(...
    'SampleTime',Ts,...
    'TargetSmoothFactor',1e-3,...
    'DiscountFactor',1.0, ...
    'MiniBatchSize',64, ...
    'ExperienceBufferLength',1e6);
agentOpts.NoiseOptions.StandardDeviation = 0.3;
agentOpts.NoiseOptions.StandardDeviationDecayRate = 1e-5;

agent = rlDDPGAgent(actor,critic,agentOpts);
maxsteps = ceil(2000);
maxepisodes = 6000;
%maxsteps = ceil(Tf/Ts);
%maxsteps = ceil(0.0001);
trainOpts = rlTrainingOptions(...
    'MaxEpisodes',maxepisodes, ...
    'MaxStepsPerEpisode',maxsteps, ...
    'ScoreAveragingWindowLength',5, ... %5
    'Verbose',false, ...
    'Plots','training-progress',...
```

```
    'StopTrainingCriteria','AverageReward',...
    'StopTrainingValue',1000,...
    'SaveAgentCriteria','EpisodeReward',...
    'SaveAgentValue',350);


%  load('Agent4632','saved_agent');
% agent = saved_agent;

%  simOpts = rlSimulationOptions('MaxSteps',maxsteps,'StopOnError','on');
%  experiences = sim(env,agent,simOpts);

%load('agent.mat','agent')
trainingStats = train(agent,env,trainOpts);
```

Appendix C

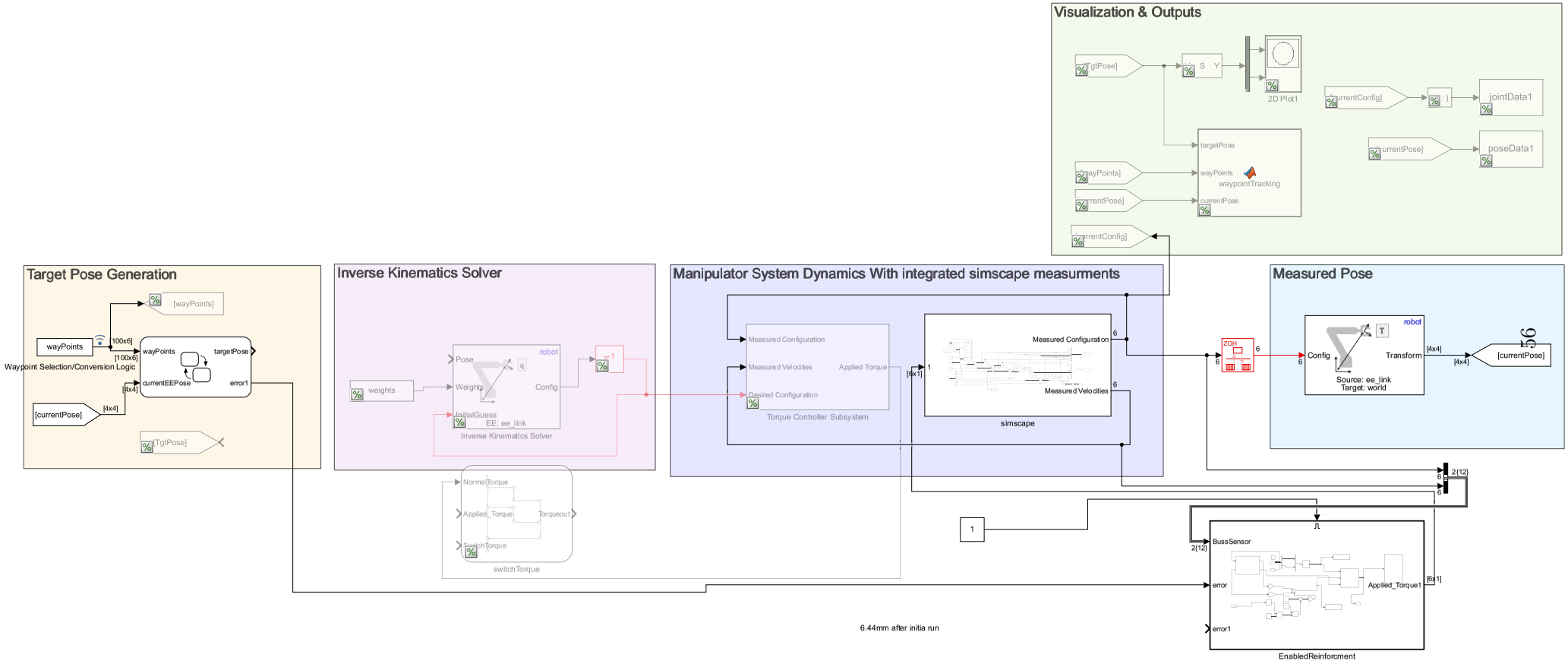
Simscape model off UR-5 Robot


1
f(x) = 0
World
world_joint
world
y-grid
Rigid Transform12
x-grid1
Rigid Transform12
base_link
shoulder_pan_joint
shoulder_link
shoulder_lift_joint
upper_arm_link
elbow_joint
forearm_link
wrist_1_joint
wrist_1_link
wrist_2_joint
wrist_2_link
wrist_3_joint
wrist_3_link
ee_fixed_joint
ee_link
velocity
position
2
Measured Velocities
1
Measured Configuration

Appendix D

Simulink Model For Training Agent

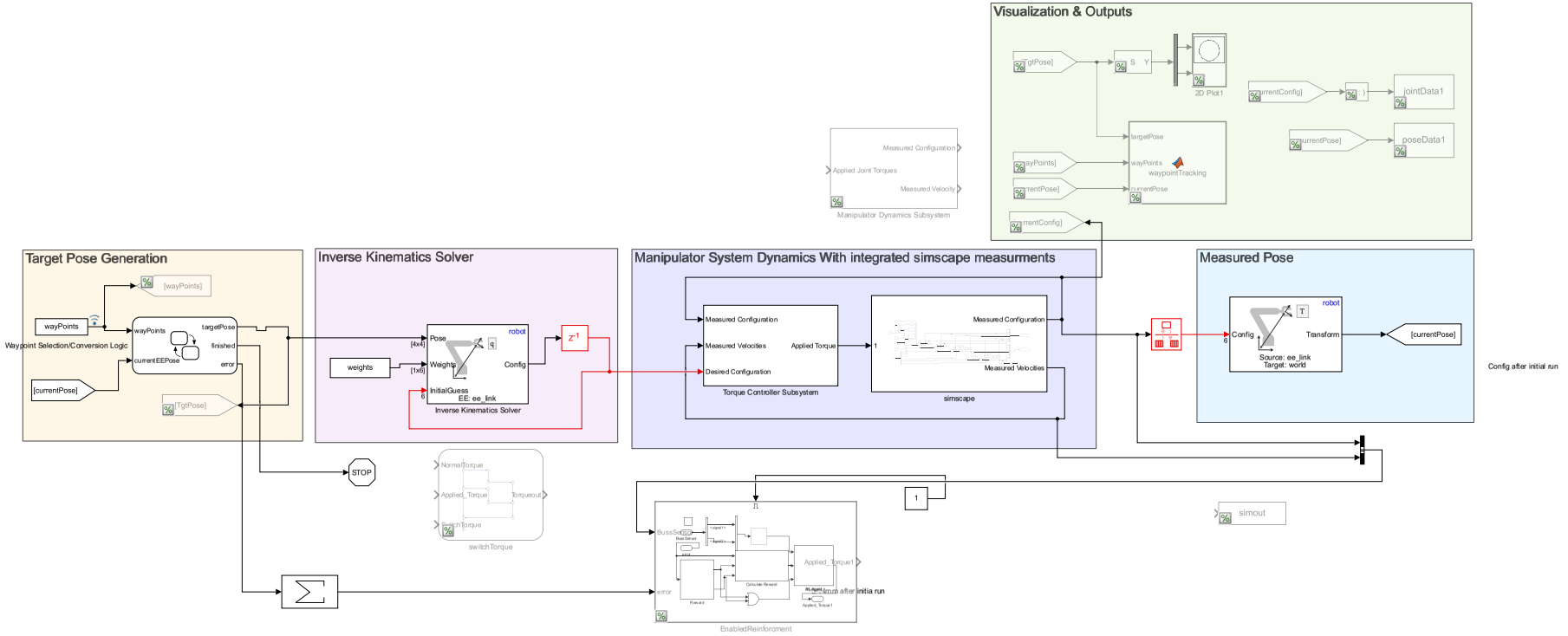
$RobotsimTrainReinforcment_{W}ay_{6}0.slxc$
https://github.com/Friggiboy/MasterRobot


Visualization & Outputs
[TgtPose]
S Y
2D Plot1
[currentConfig]
:)
jointData1
targetPose
wayPoints
waypointTracking
currentPose
[currentPose]
poseData1
[wayPoints]
[currentConfig]
Target Pose Generation
[wayPoints]
wayPoints
100x6
[100x6]
Waypoint Selection/Conversion Logic
wayPoints
targetPose
currentEEPose
error1
[4x4]
[currentPose]
[4x4]
[TgtPose]
Inverse Kinematics Solver
weights
Pose
Weights
robot
q
Config
InitialGuess
EE: ee_link
Inverse Kinematics Solver
1
switchTorque
NormalTorque
Applied_Torque
Torqueout
SwitchTorque
switchTorque
Manipulator System Dynamics With integrated simscape measurments
Measured Configuration
Measured Velocities
Applied Torque
Desired Configuration
Torque Controller Subsystem
Measured Configuration
Measured Velocities
simscape
6
1
[6x1]
ZOH
6
Measured Pose
robot
T
Config
Transform
[4x4]
[4x4]
Source: ee_link
Target: world
[currentPose]
2{12}
6
1
1
BussSensor
2{12}
error
error1
Applied_Torque1
[6x1]
EnabledReinforcment
6.44mm after initia run

Appendix E

Simulink Model Conventional Control

The files for runing the script can be found here: https://github.com/Friggiboy/MasterRobot
The file: "LocationOfBoltAndWaypointGen.m" Much be runed befor the file "robotsimConventinalControl2405.slx "


Visualization & Outputs
tgtPose]
S
Y
2D Plot1
currentConfig]
1
jointData1
currentPose]
poseData1
targetPose
wayPoints
waypointTracking
currentPose
ayPoints]
rrentPose]
rrentConfig]
Measured Configuration
Applied Joint Torques
Measured Velocity
Manipulator Dynamics Subsystem
Target Pose Generation
[wayPoints]
wayPoints
Waypoint Selection/Conversion Logic
wayPoints
currentEEPose
targetPose
finished
error
[currentPose]
[TgtPose]
Inverse Kinematics Solver
weights
[4x4]
[1x6]
Pose
Weights
InitialGuess
6
robot
q
Config
EE: ee_link
Inverse Kinematics Solver
z-1
Manipulator System Dynamics With integrated simscape measurments
Measured Configuration
Measured Velocities
Desired Configuration
Applied Torque
Torque Controller Subsystem
1
Measured Configuration
Measured Velocities
simscape
Measured Pose
robot
T
Config
6
Transform
Source: ee_link
Target: world
[currentPose]
Config after initial run
STOP
NormalTorque
Applied_Torque
SwitchTorque
Torqueout
switchTorque
1
simout
BussSelector
Base Servo
error
Reward
Calculate Reward
RL Agent
Applied_Torque1
Torque after initia run
EnabledReinforcment

Appendix F

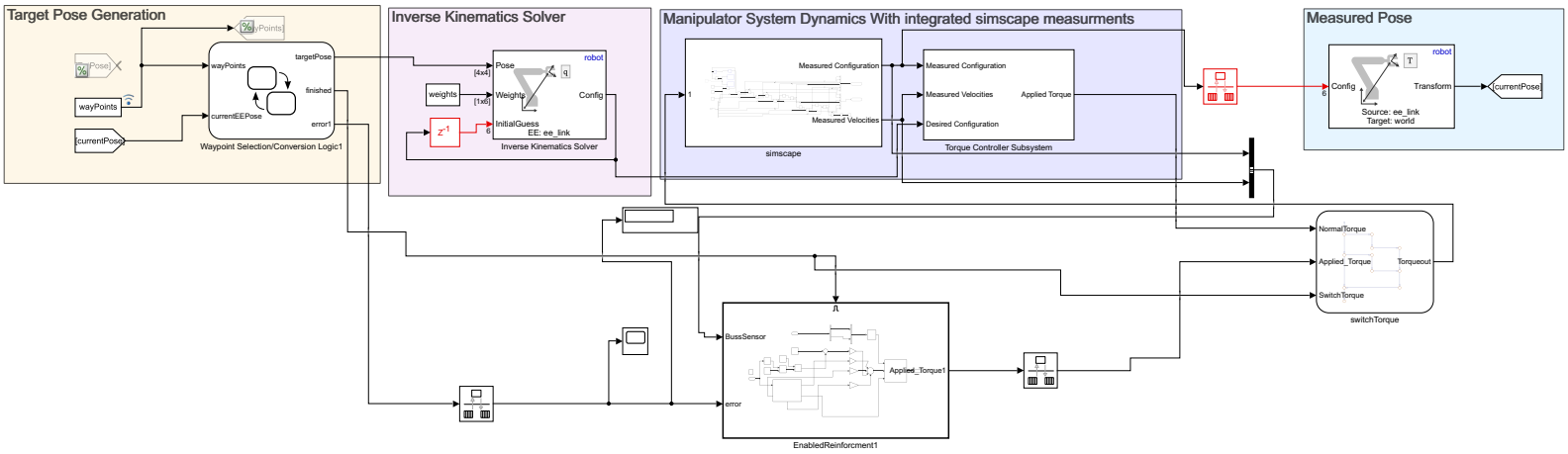
Simulink Model Combination of Conventional Control And Reinforcement Control

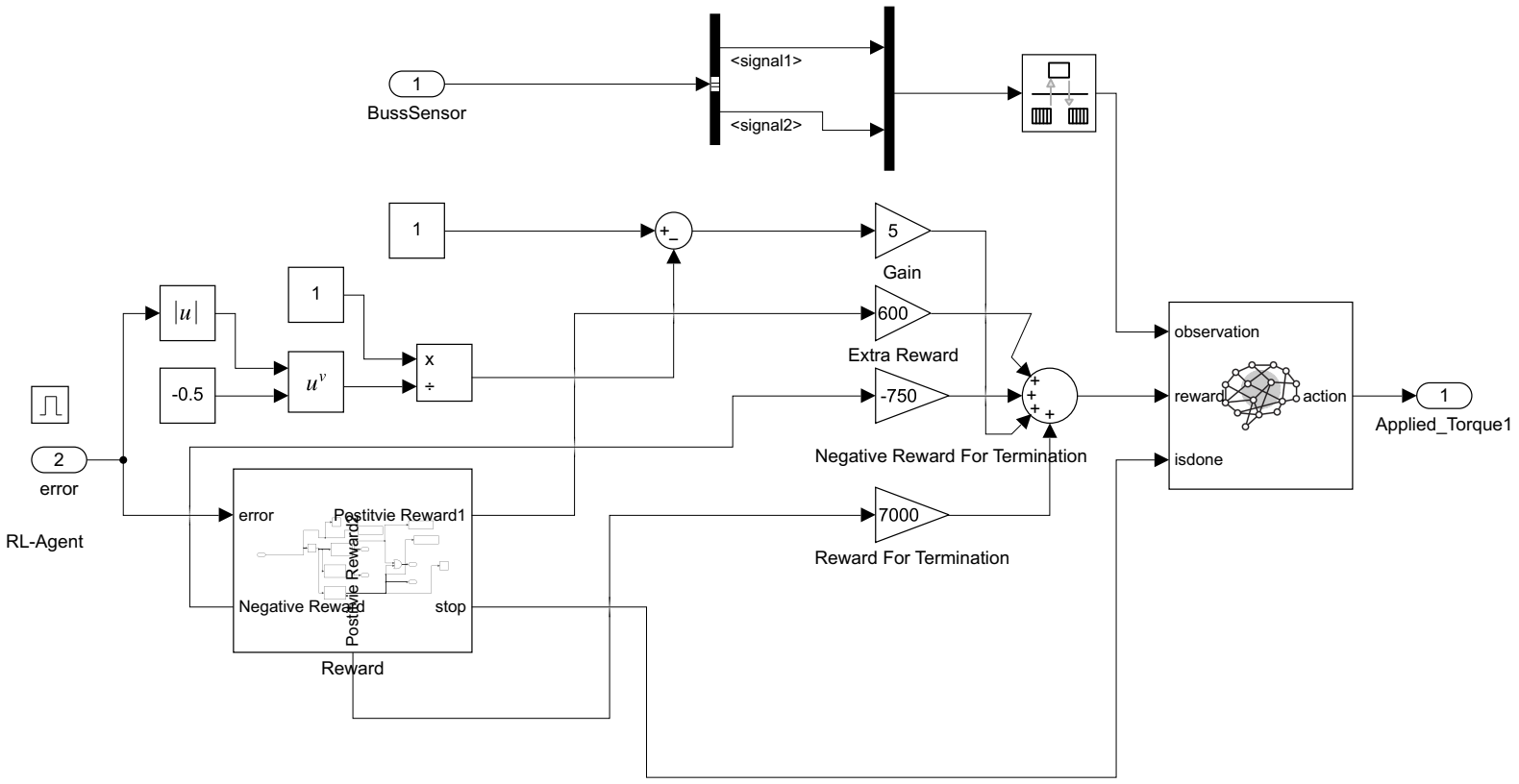
Link to youtube video of simulation:

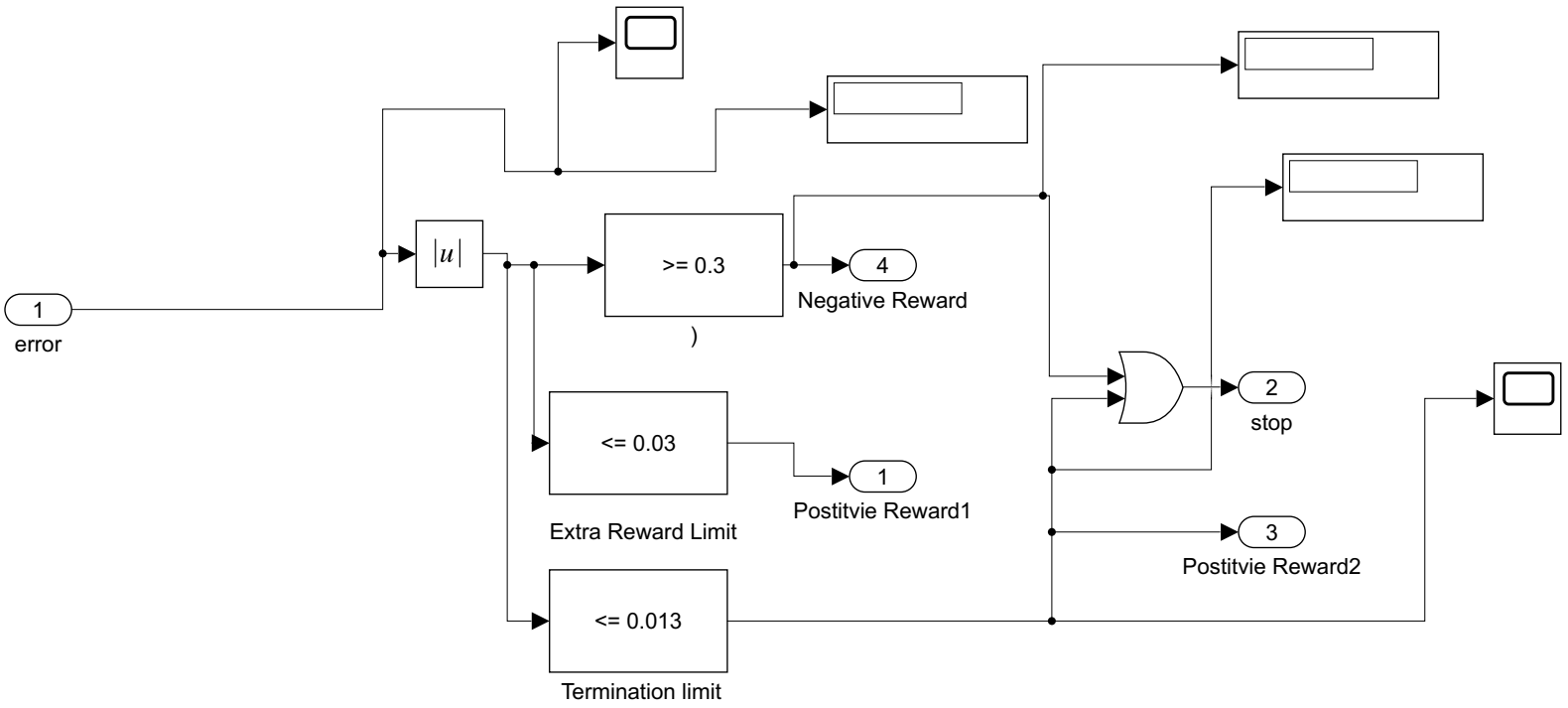
https://studio.youtube.com/video/tk0Aq-JXFf0/edit

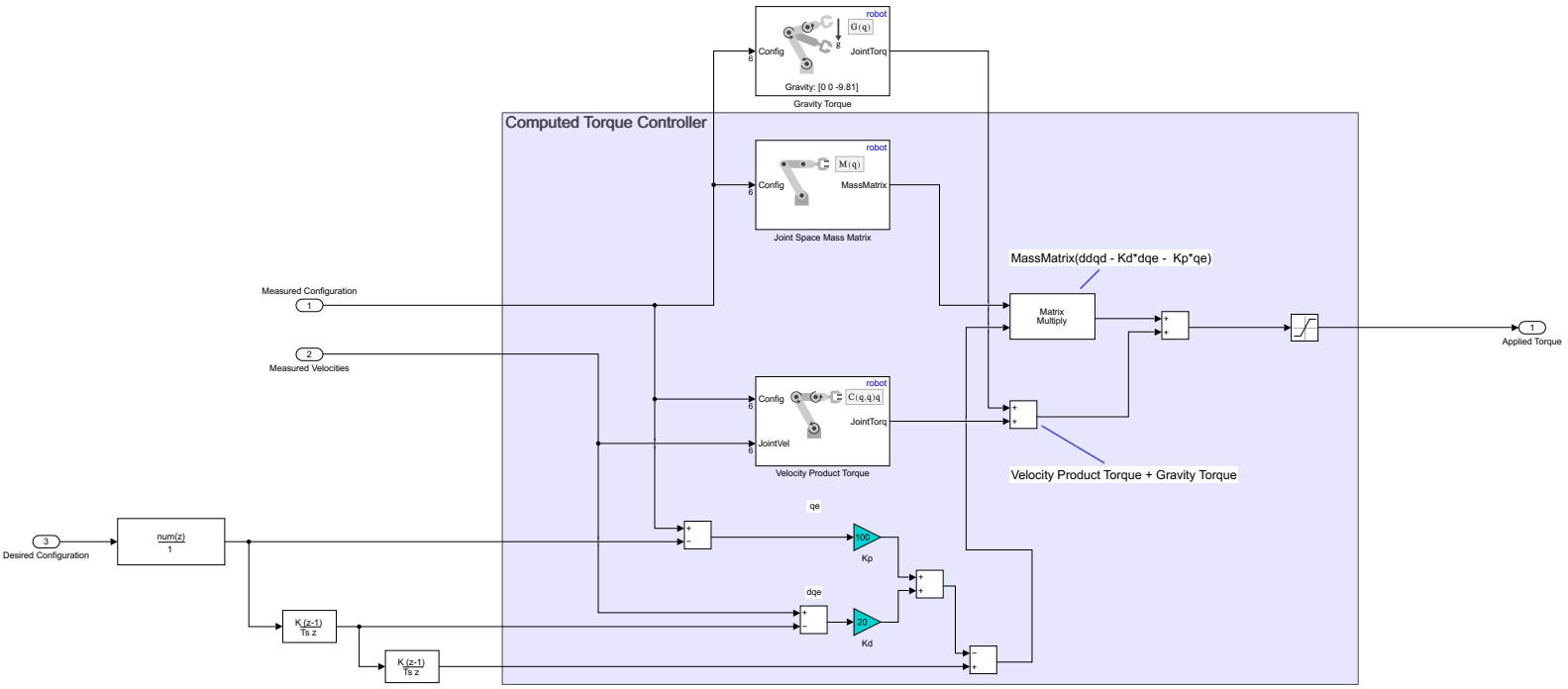
$https://www.youtube.com/watch?v=8DEd-2_lfzQ$

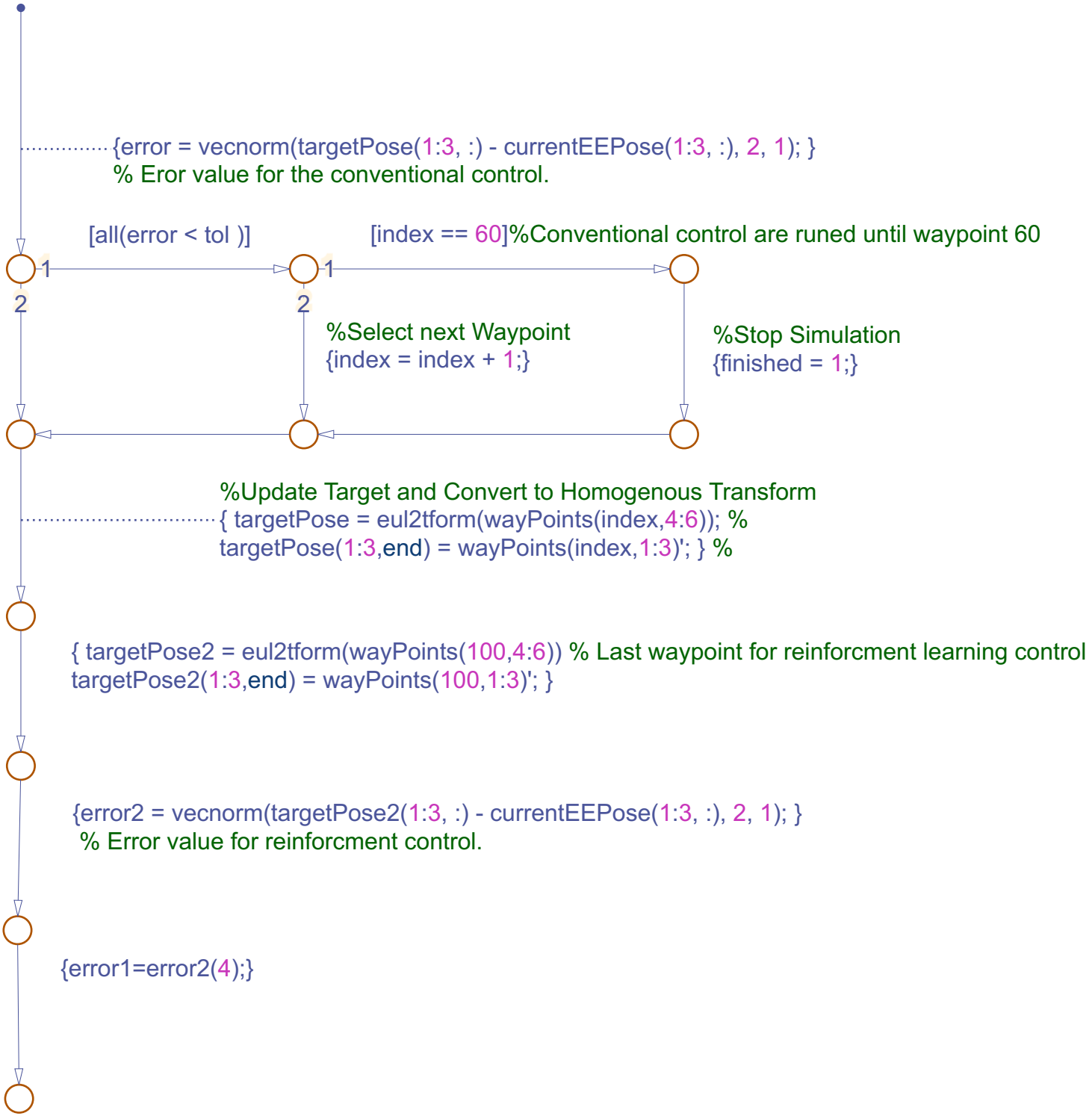
Link to matlab script and simulink files
https://github.com/Friggiboy/MasterRobot (Agent.mat files are to big to upload, but a link to fileconvoy is added)

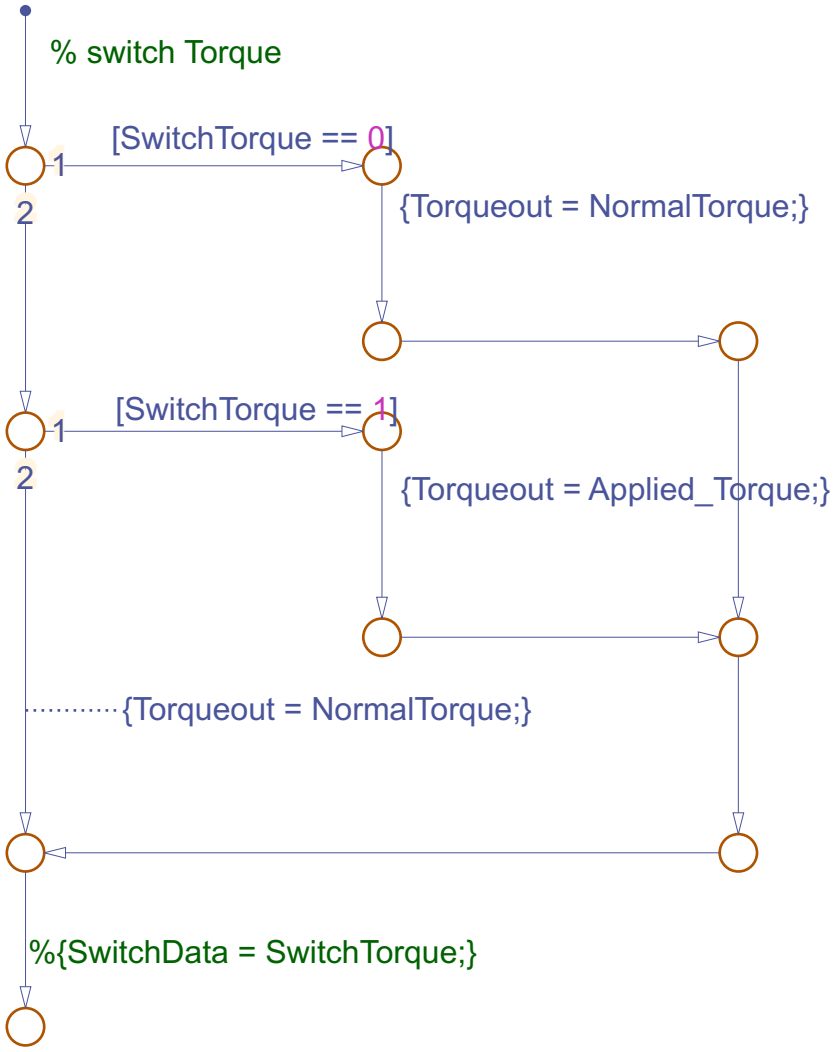

Target Pose Generation
wayPoints
wayPoints
currentPose
wayPoints
currentEEPose
targetPose
finished
error1
Waypoint Selection/Conversion Logic1
Inverse Kinematics Solver
Pose
[4x4]
weights
[1x6]
InitialGuess
robot
Weights
Config
EE: ee_link
Inverse Kinematics Solver
Manipulator System Dynamics With integrated simscape measurments
Measured Configuration
Measured Velocities
simscape
Measured Configuration
Measured Velocities
Desired Configuration
Applied Torque
Torque Controller Subsystem
Measured Pose
robot
Config
Transform
currentPose
Source: ee_link
Target: world
NormalTorque
Applied_Torque
SwitchTorque
Torqueout
switchTorque
BussSensor
error
Applied_Torque1
EnabledReinforcment1


1
BussSensor
<signal1>
<signal2>
1
5
Gain
600
Extra Reward
-750
Negative Reward For Termination
7000
Reward For Termination
|u|
1
-0.5
u^v
x
÷
observation
reward
action
isdone
1
Applied_Torque1
2
error
RL-Agent
error
Pestitvie Reward1
Negative Reward
stop
Postive Reward
Reward


1
error
|u|
>= 0.3
4
Negative Reward
)
<= 0.03
1
Postitvie Reward1
Extra Reward Limit
<= 0.013
Termination limit
2
stop
3
Postitvie Reward2


Computed Torque Controller
robot
G(q)
Config
JointTorq
g
Gravity: [0 0 -9.81]
Gravity Torque
robot
M(q)
Config
MassMatrix
Joint Space Mass Matrix
MassMatrix(ddqd - Kd*dqe - Kp*qe)
Measured Configuration
1
2
Measured Velocities
Matrix
Multiply
1
Applied Torque
robot
C(q,q̇)q̇
Config
JointTorq
JointVel
Velocity Product Torque
Velocity Product Torque + Gravity Torque
qe
100
Kp
dqe
20
Kd
3
Desired Configuration
num(z)
1
K (z-1)
Ts z
K (z-1)
Ts z


{error = vecnorm(targetPose(1:3, :) - currentEEPose(1:3, :), 2, 1); }
% Eror value for the conventional control.
[all(error < tol)]
[index == 60]%Conventional control are runed until waypoint 60
1
2
1
2
%Select next Waypoint
{index = index + 1;}
%Stop Simulation
{finished = 1;}
%Update Target and Convert to Homogenous Transform
{ targetPose = eul2tform(wayPoints(index,4:6)); %
targetPose(1:3,end) = wayPoints(index,1:3)'; } %
{ targetPose2 = eul2tform(wayPoints(100,4:6)) % Last waypoint for reinforcment learning control
targetPose2(1:3,end) = wayPoints(100,1:3)'; }
{error2 = vecnorm(targetPose2(1:3, :) - currentEEPose(1:3, :), 2, 1); }
% Error value for reinforcment control.
{error1=error2(4);}


% switch Torque
[SwitchTorque == 0]
1
2
{Torqueout = NormalTorque;}
[SwitchTorque == 1]
1
2
{Torqueout = Applied_Torque;}
{Torqueout = NormalTorque;}
%{SwitchData = SwitchTorque;}

Bibliography

[1] Bonsai. *Writing Great Reward Functions*. URL: https://www.youtube.com/watch?v=0R3PnJEisqkhttps://www.youtube.com/watch?v=0R3PnJEisqk. (accessed: 04.06.2021 09:47/19:25).

[2] Frank C. Park Kevin M. Lynch. *Modern Robotics*. Cambridge University Press, 2017. Chap. 2.2.

[3] Frank C. Park Kevin M. Lynch. *Modern Robotics*. Cambridge University Press, 2017. Chap. 4.

[4] Frank C. Park Kevin M. Lynch. *Modern Robotics*. Cambridge University Press, 2017. Chap. 3.3.1.

[5] Mathworks. *Boundary Tracing in Images*. URL: https://se.mathworks.com/help/images/boundary-tracing-in-images.html. (accessed: 02.05.2021).

[6] Mathworks. *Create Simulink Environment and Train Agent*. URL: https://se.mathworks.com/help/reinforcement-learning/ug/create-simulink-environment-and-train-agent.html. (accessed: 01.03.2021).

[7] Mathworks. *Identifying Round Objects*. URL: https://se.mathworks.com/help/images/identifying-round-objects.html. (accessed: 29.04.2021).

[8] Mathworks. *Machine learning with MATLAB*. URL: https://se.mathworks.com/content/dam/mathworks/ebook/gated/machine-learning-ebook-all-chapters.pdf. (accessed: 03.04.2021).

[9] Mathworks. *Trajectory Control Modeling with Inverse Kinematics*. URL: https://se.mathworks.com/help/robotics/ug/trajectory-control-modeling-with-inverse-kinematics.html. (accessed: 02.02.2021).

[10] Timothy Lillicrap. Jonathan Hunt. Alexander Pritzel. Nicolas Heess. Tom Erez. Yuval Tassa. David Silver and Daan Wierstra. *Continuous Control with Deep Reinforcement Learning*. URL: https://arxiv.org/abs/1509.02971. (accessed: 01.03.2021).

[11] Pablo Quílez Velilla. *URDF file for ur5 robot*. URL: https://github.com/ros-industrial/robot_movement_interface/blob/master/dependencies/ur_description/urdf/ur5_robot.urdf. (accessed: 17.02.2021).