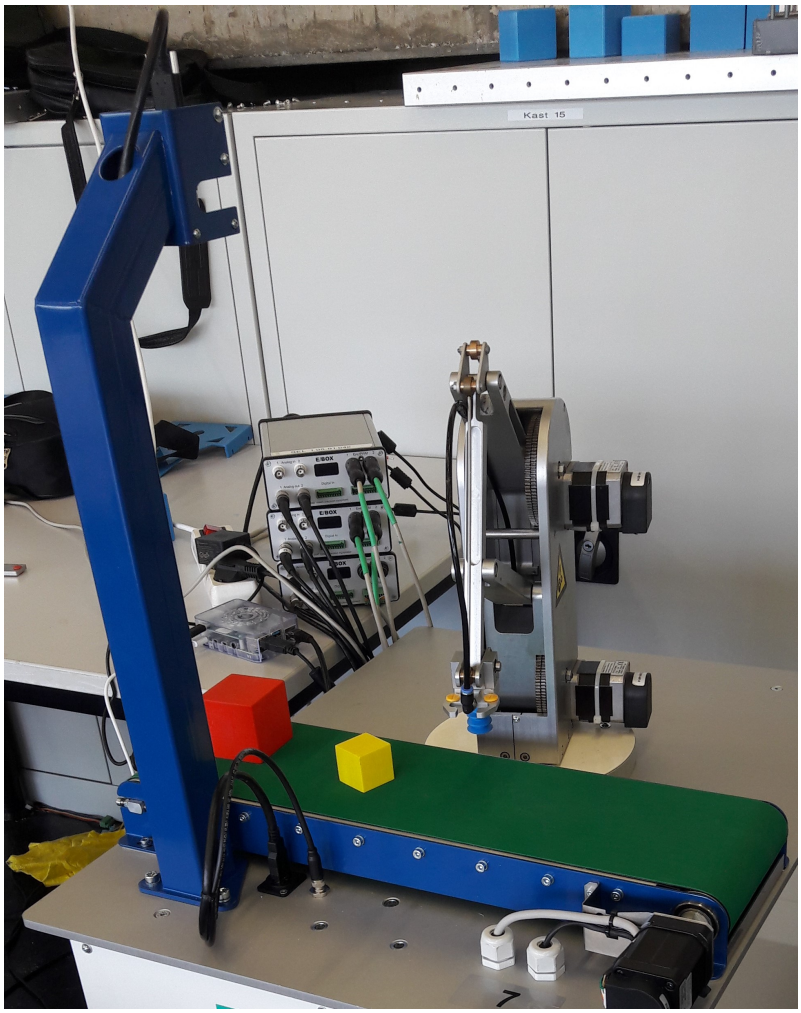
EINDHOVEN UNIVERSITY OF TECHNOLOGY

4GB20 Robot-arm Manual



R.E.W.L. Thijssen
Q. Swaan
G.P.M. van Hattuum

January 31, 2025

Contents

1 System overview

The robot arm setup consists of several parts, seen in Figure 1: the robot arm (1), E-boxes (2), camera (3), conveyor belt (4) and Raspberry Pi (not shown in the figure). Your laptop can directly communicate with the camera, however communicating with the robot arm and conveyor belt is more involved. This requires the E-boxes, Raspberry Pi and SPERTE. The E-boxes are ethercat measurement boxes that change digital signals of your laptop to analogue signals for the setup and vice versa. The E-boxes solely work with Linux, whereas your laptop does not necessarily run Linux. Therefore, the Raspberry Pi is used to communicate with the E-boxes. Lastly, the SPERTE software is used to convert the MATLAB code and models of your laptop to code that the Raspberry Pi can understand.
The camera and robot arm itself work in the same coordinate frame, which is defined by the inverse kinematics. The origin of these coordinates is located in the center of the cylindrical pedestal (on which the robot arm stands) at the height of large rectangular tabletop. The X-axis is perpendicular to the direction of the conveyor belt, whereas the Y-axis is parallel. This can be seen in Figure 2. The Z-axis is pointing upwards to create a right-handed coordinates.
Before every run, the robot arm needs to home. The goal of this homing is be sure that it starts at the same position every run. That position is defined as such that the tip of the vacuum gripper is at $\begin{bmatrix} X & Y & Z \end{bmatrix} = \begin{bmatrix} 160 & 0 & 99 \end{bmatrix}$.

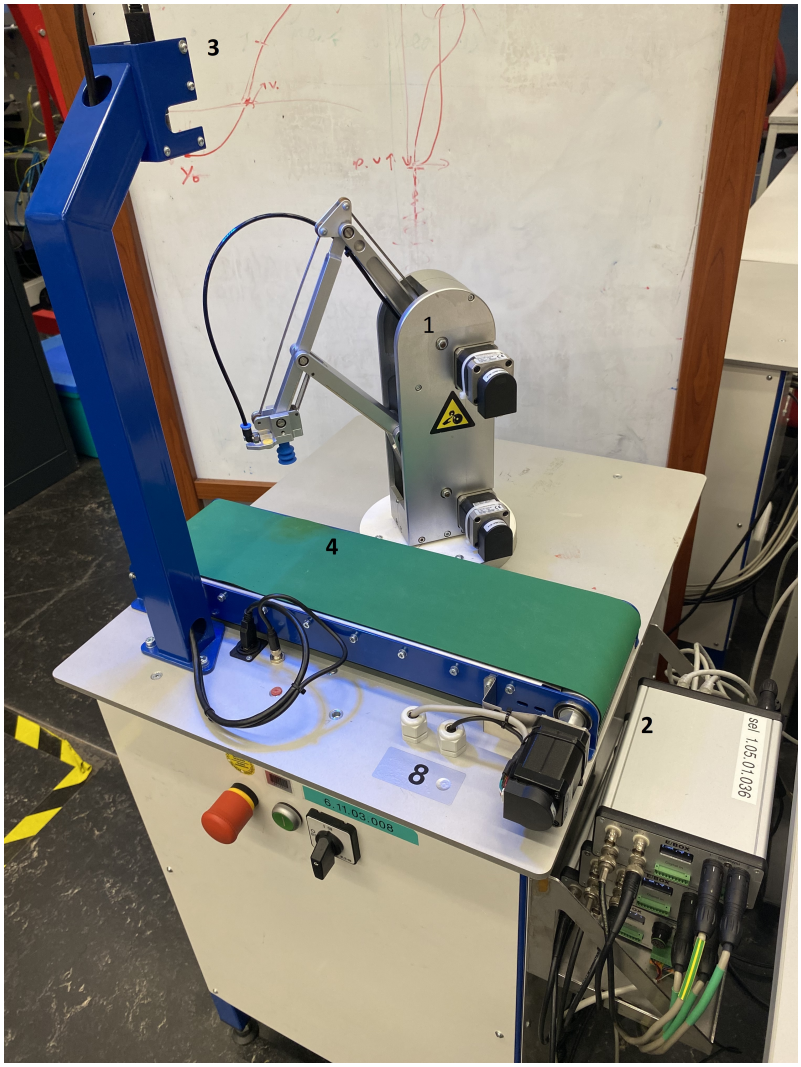


Figure 1: Robot arm setup. (1) robot arm, (2) E-boxes, (3) camera and (4) conveyor belt.

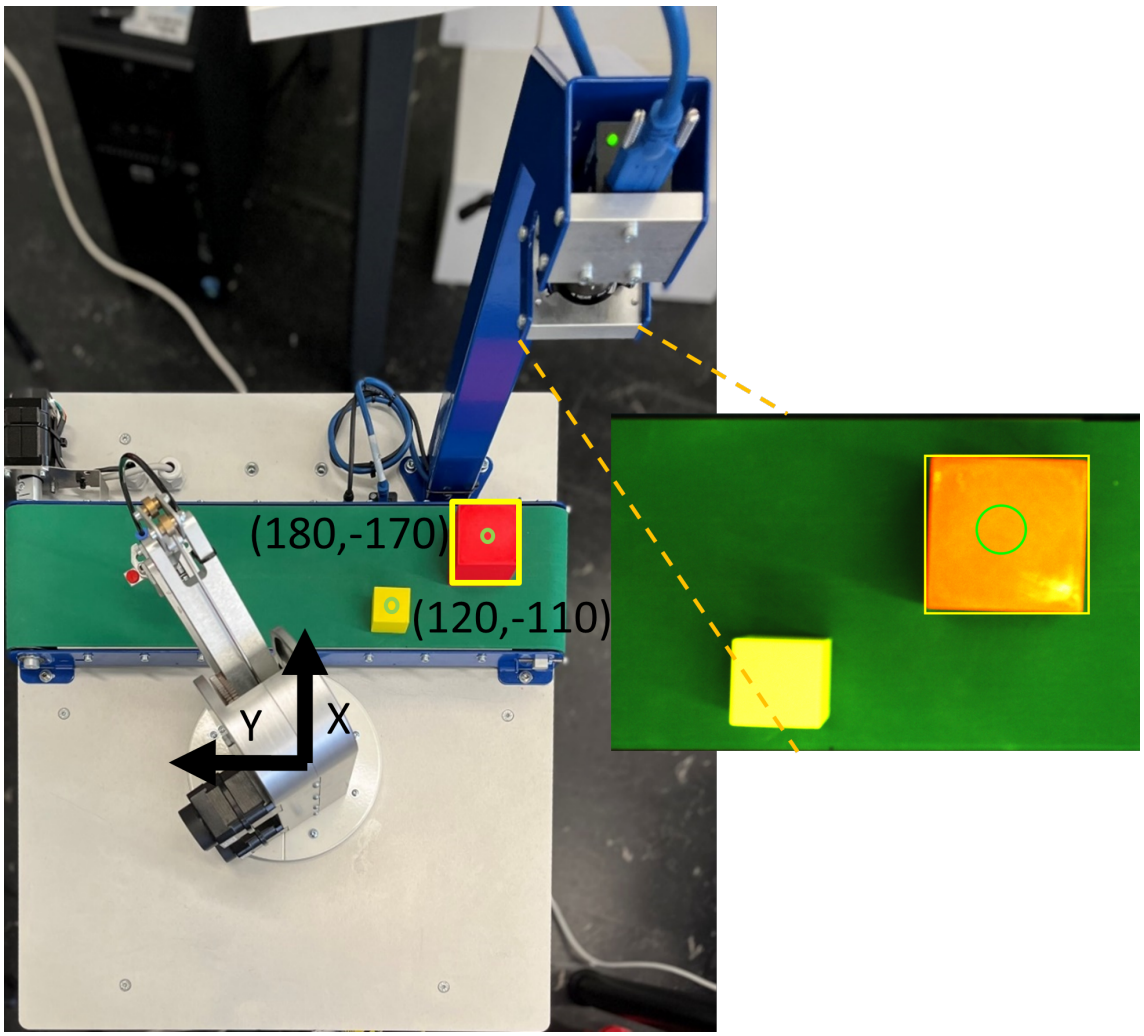


Figure 2: Example of XY-coordinates.

2 Installations

This section describes the different installations that have to be done prior to working with the setup. Hence it is only required to do this once. Before installing, first download and unzip the file `robotarm_student.zip` from Canvas. This file includes everything you need to install and run the robot-arm. Furthermore, MATLAB version 2021a is used in this course. Please download this specific version to avoid version problems with the robot-arm software and with your fellow students' software. The camera does not support macOS. All other parts of the system do work with macOS. So, when using macOS, measuring FRF's and controlling the robot is possible, but running the camera is not. macOS might block some robot arm software as the developer is not identified. Please allow this software to run anyway in the security settings.

2.1 SPERTE

In order to install the SPERTE software, follow the steps in `SPERTE_Installation.pdf`. Skip the paragraph 'Installing the Sperte Tooling and Examples'.

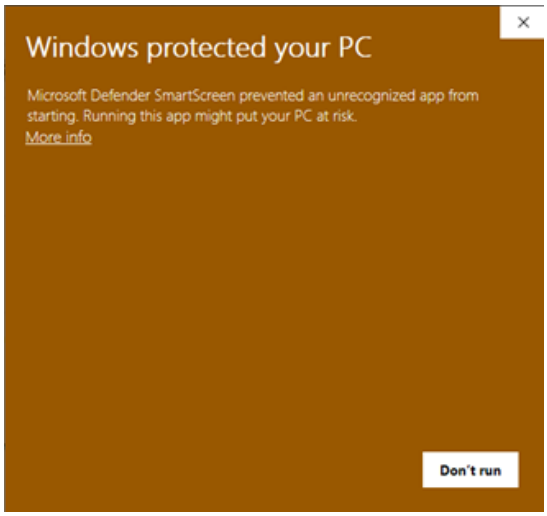
2.2 Toolboxes

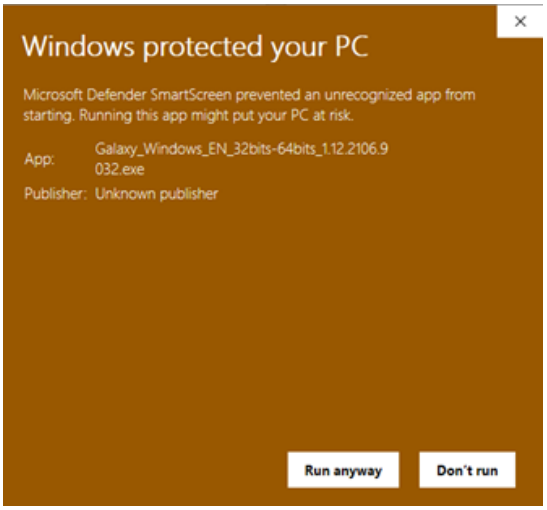
In order for the camera to function properly the following toolboxes must be downloaded from the MATLAB Add-on explorer. To find and install add-ons, go to the Home tab and, in the Environment section, click the Add-Ons icon.

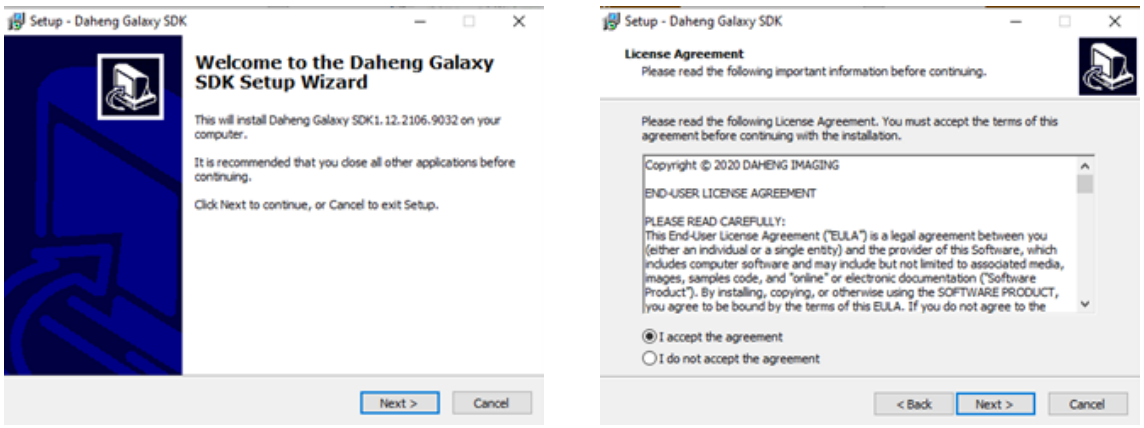
- Image Acquisition Toolbox
- Image Acquisition Toolbox Support Package for OS Generic Video Interface
- Computer Vision Toolbox
- Image Processing Toolbox
- Stateflow

2.3 Camera software

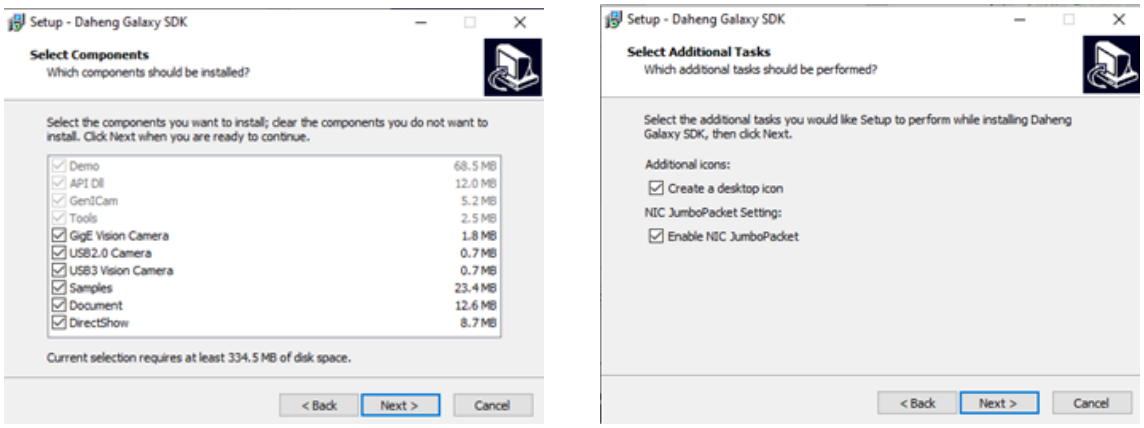
The camera drivers can be installed by running `Galaxy_Windows_EN_32bits-64bits_1.12.2106.9032.exe` and following the steps underneath.







Choose right folder



Check all boxes

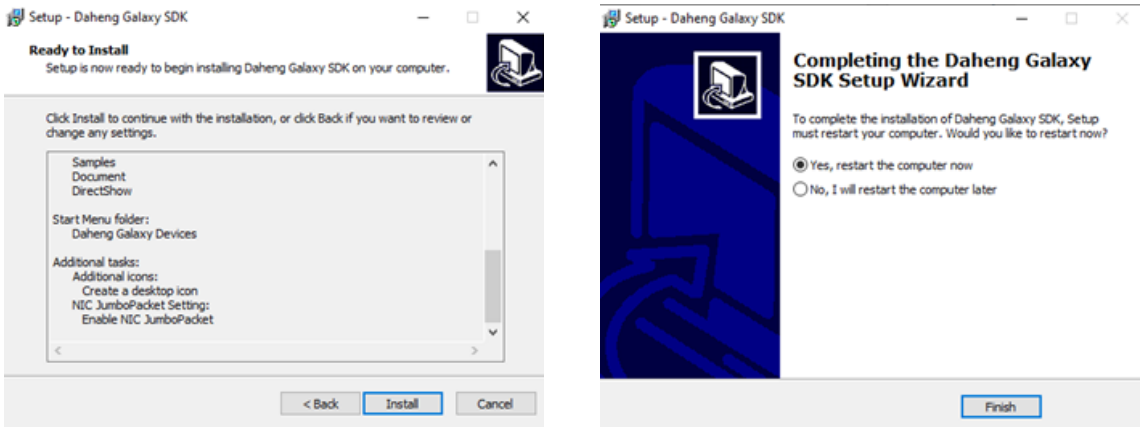


Figure 3: Installation steps for the camera drivers

2.4 Robot arm

Please run the MATLAB file `Install.m` to install a custom Simulink library, RTScope, Sperte and other functions in order to run the robot arm properly.
To check if all toolboxes and addons are installed, run the file CheckAddons.m. This file could be found in Robotarm_student/tooling.

2.5 ShapeIt

ShapeIt is convenient tool, which can be used to develop controllers. To install ShapeIt follow these steps.

3 Raspberry Pi

In this section it is explained how to connect to the Raspberry Pi with your laptop. Three different approaches are covered. Usually the default method suffices. In case this does not work, you can try the other two methods.

3.1 Default method

The default method is to connect to the Raspberry Pi over USB. A cabled connection is stable and easy to configure.

1. Get a USB-C cable, an Ethernet cable and a Raspberry Pi power adapter from the cart or from SEL/SOL. Connect the top USB-C port of the Pi to power, and the bottom USB-C port to your computer.
2. Wait for at least 45 seconds for the Pi to boot after plugging in the power.

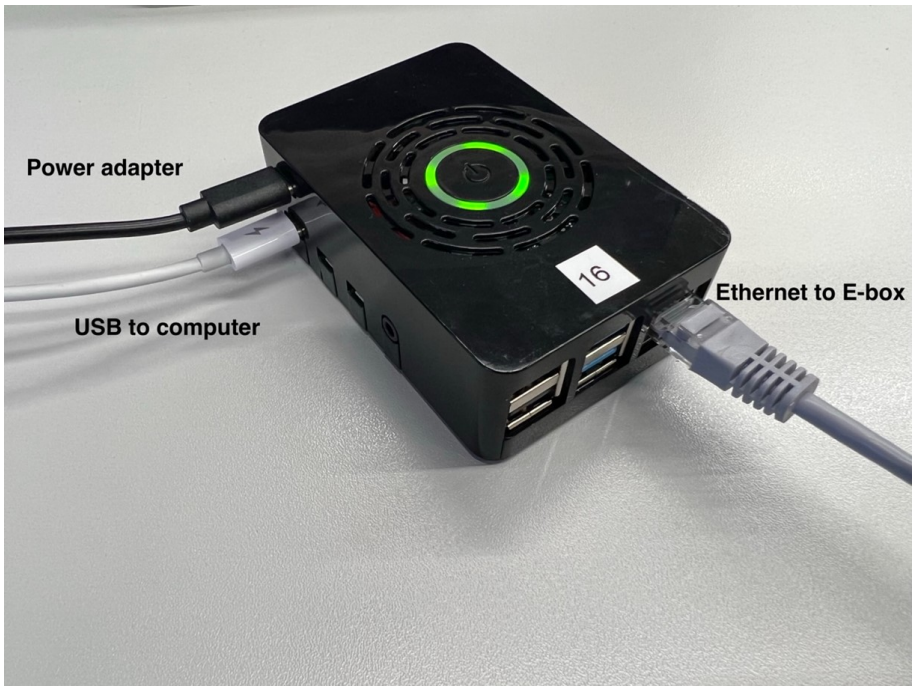


Figure 4: Connetion Raspberry Pi with USB-C

3. Enter in the MATLAB command window: p = raspi('10.55.0.1','pi','sperte123').
   If you entered this line correctly but you still get an error, follow these instructions carefully to install the correct drivers on your computer. Reboot the Pi after installing the drivers by pressing the power button once, waiting for 10 seconds, unplugging both usb cables, and reconnecting both cables. If you can still not connect, proceed to an alternative method.
4. When you're done experimenting, press the power button once and wait for it to turn red before disconnecting the power.

3.2 Alternative method 1

Another possibility is to connect to the Raspberry Pi via an ethernet connection.

1. Get a Raspberry Pi power adapter, an USB-A to Ethernet adapter and two Ethernet cables from the cart or from SEL/SOL. Connect the cables as shown above, making sure that the power cable is connected to the TOP usb-c port on the side of the Pi.

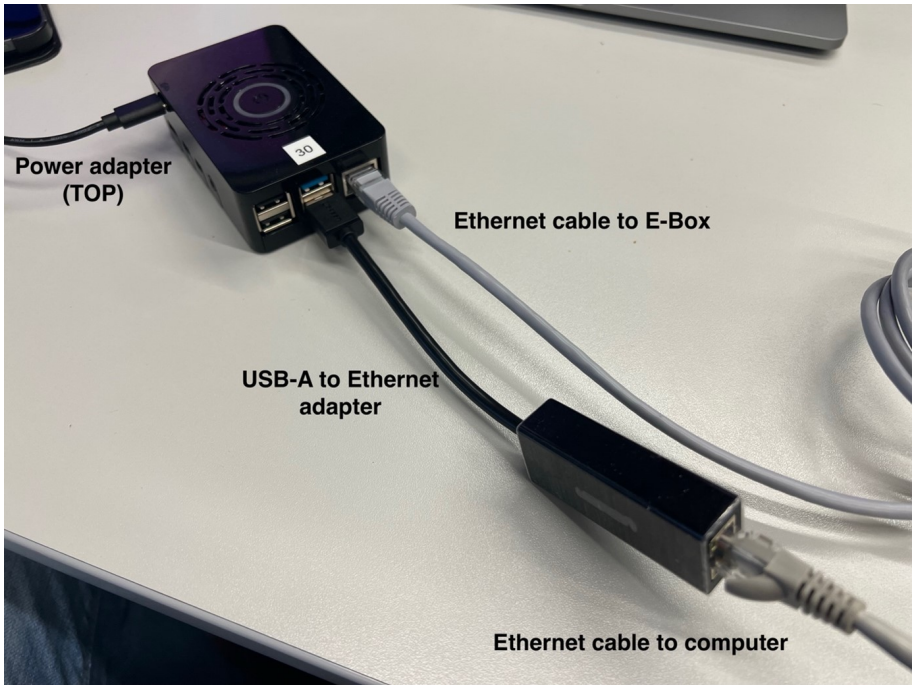


Figure 5: Connection Raspberry Pi with Ethernet cable.

2. Wait for at least 45 seconds for the Pi to boot after plugging in the power.
3. Configure the Ethernet adapter in your computer to use a static IP address, as follows:
   (a) On Windows: right click your Windows button in the bottom left corner, select "Windows Powershell (Admin)" and enter: netsh interface ip set address name = "Ethernet" static 192.168.137.9 255.255.255.0 If this throws an error, your Ethernet adapter might have a different name than "Ethernet". In that case, look through the output of ipconfig /all in Powershell to find out the name of your Ethernet adapter and replace it in the command above. When you want to use your Ethernet port for internet again after experiments, run the following command to reconfigure your Ethernet adapter for automatic IP addresses: netsh interface ip set address name = "Ethernet" dhcp If you prefer to follow these steps graphically, as opposed to using the above commands, follow this guide and make sure you use IP 192.168.137.9 with subnet mask 255.255.255.0.
   (b) On Mac: follow this guide and make sure you use IP 192.168.137.9 with subnet mask 255.255.255.0, instead of the IP address used in the guide. If you are using an Ethernet dongle for your Mac, make sure to select the correct Ethernet adapter in the network settings. When you want to use your Ethernet port for internet again after experiments, change the settings back to automatic (DHCP).
   (c) On Linux, Google how to do this for your specific Linux distribution.
4. Enter in the MATLAB command window: p = raspi('10.55.0.1','pi','sperte123').
5. When you're done experimenting, press the power button once and wait for it to turn red before disconnecting the power.

3.3 Alternative method 2

The last option is to connect over wifi. Although a wireless connection is less stable than a wired connection, and occasionally one might find that the data contains odd jumps or other distortions, this is a viable and easy way to connect if the previous two methods failed.

1. Make sure your laptop is connected to tue-wpa2.
2. Get a Raspberry Pi power adapter and an Ethernet cable from the cart or from SEL/SOL. Connect the power adapter to the TOP usb-c port on the side of the Pi and press the power button once so it turns green. Connect the Ethernet cable to the E-box.

Figure 6: Connention Raspberry Pi with wifi.

3. Wait for at least 45 seconds for the Pi to boot after plugging in the power.
4. Enter in the MATLAB command window: p = raspi('SPERTEx.bwk.tue.nl','pi','sperte123'), where you replace x with the number written on your Pi (1 would become SPERTE1 and 12 would become SPERTE12).
5. When you're done experimenting, press the power button once and wait for it to turn red before disconnecting the power.

4 Control robot

This section describes the steps to connect with and control the robot arm and camera.
Before performing any of these steps please make sure to have finished all steps in section 2.
To prevent crashes, it is important to follow all steps in the correct order.

1. Open MATLAB.
2. Connect the Raspberry Pi to your laptop according to section 3.
3. Plug in the camera to a USB port of your laptop. Preferably a high power USB port (indicated by a lightning bolt next to the USB port) to avoid the camera going into low-power mode.
4. Open Simulink script `robotarm_student_2021a_v2.slx`.
5. Optionally, rename the Simulink model in order to backup the initial model.
6. Initialize the model by pressing *Initialize* in the top left of the Simulink model.
7. Fill in the robot number.
8. Optionally, make changes to the Simulink model.
9. Under the tab Hardware, in the section Run on hardware, click *Monitor & Tune*.
10. Wait for the model to build and the robot arm to home. This may take a minute, particularly the first time.
11. Optionally, run the camera by running `camera.m` in MATLAB as described in section 5.
12. Adjust parameters in the Simulink model to control the setup as described in section 6.
13. When done, stop the execution by clicking *Stop* that appeared in place of *Monitor & Tune*.

After doing this for the first time, return to item 8 if you are making changes to the current model.
Or return to item 5 if you are renaming the model.

5 Camera

In this section it is explained how to use the camera together with Simulink to detect objects as well as how to make changes to the object detection.

5.1 Perform object detection

The object detection and communication is done the `camera.m` script. This script receives pictures from the camera and tries to find connected areas of certain colors. It can detect three different objects, each within a specific color range (mask). For every color you can also choose a specific area range. This range is expressed in number of pixels. In the script there are already three default objects given. The area of each object can directly be changed in the script, but changing the mask will be explained in subsection 5.2.
When Simulink files are becoming too complex or large, the Raspberry Pi cannot keep up with the computations anymore. The result of this is that Simulink starts lagging. The first option to fix this is to delete unnecessarily heavy features such as a lot of scopes from your script. If this is not sufficient, you can try to decrease the rate at which the object detection is performed. This can be done in the `camera.m` script. Be wary that a higher detection rate than 5 Hz will cause Simulink to start lagging, for similar reasons as mentioned above. However, a too low detection rate could cause the camera to miss objects that are going too fast.
The remainder of the script handles the communication and object detection. There is no need for changing this.

5.2 Change object detection

If it appears that the default masks are not sufficient for your particular object, it is possible to change these using the following method.

1. Open `camera_standalone.m`
   This script allows to use the camera without connecting to Simulink. This opens the live camera feed, as displayed in Figure 7. The window also adds a bounding box around the objects. This is the rectangle the encloses each object. ON top of this, it shows the centroid. The centroid is located in middle of each object and roughly has the size of the vacuum gripper. Also, the area of each object is displayed in green.

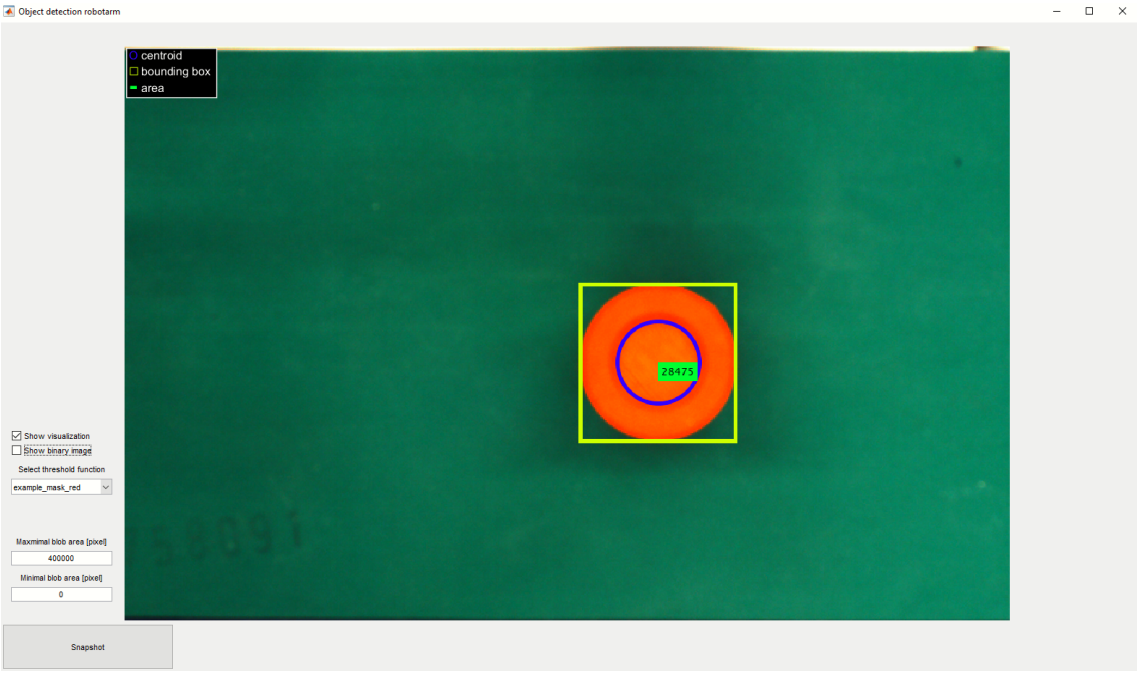


Figure 7: Live feed of camera within `camera_standalone.m`.

On the left you can use the dropdown menu to select the color mask that is applied to the feed on the right. Underneath the dropdown menu, the area range for the object detection

in the feed can be adjusted. Note that the chosen area ranges must be manually changed in the `camera.m` script.
Above the dropdown menu are two toggle switches for the visualisations and the binary image. The binary image, as displayed in Figure 8, essentially shows the color mask. The pixels that are within the color range are displayed in white, whereas the other pixels are black.

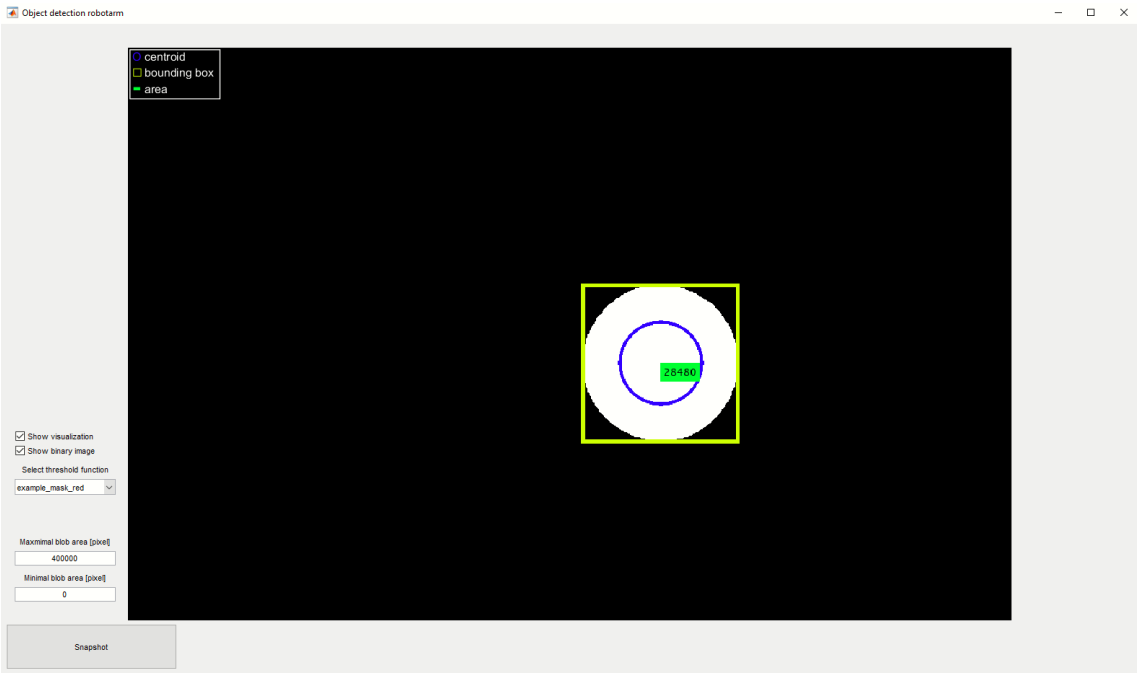


Figure 8: Binary image of the camera live feed within `camera_standalone.m`.

2. Take snapshot
   Press snapshot on the bottom left to save a snapshot of the current camera feed. In the popped up window choose a file name and save the figure.

3. Close the live feed window
   This should also automatically stop the `camera_standalone.m` script.

4. Open the Color Thresholder app
   Either find this application in the *Apps* tab or type `colorThresholder` in the command window.

5. Load image
   Press *Load Image* on the top left and select the snapshot you just saved.

6. Choose a color space
   Select one of the color spaces that are offered. *RGB* allows you to select red, green and blue ranges for your mask. This is quite intuitive, however other color spaces represent the colors in a different way, which can be convenient as well. Information on these color spaces is readily available online.

7. Tune mask
   Now you may adjust the color ranges on the top right to create the mask for the object detection. For example, in Figure 9 the pixels with a certain level of blue are ignored, which blocks out the conveyor belt, leaving the red object.

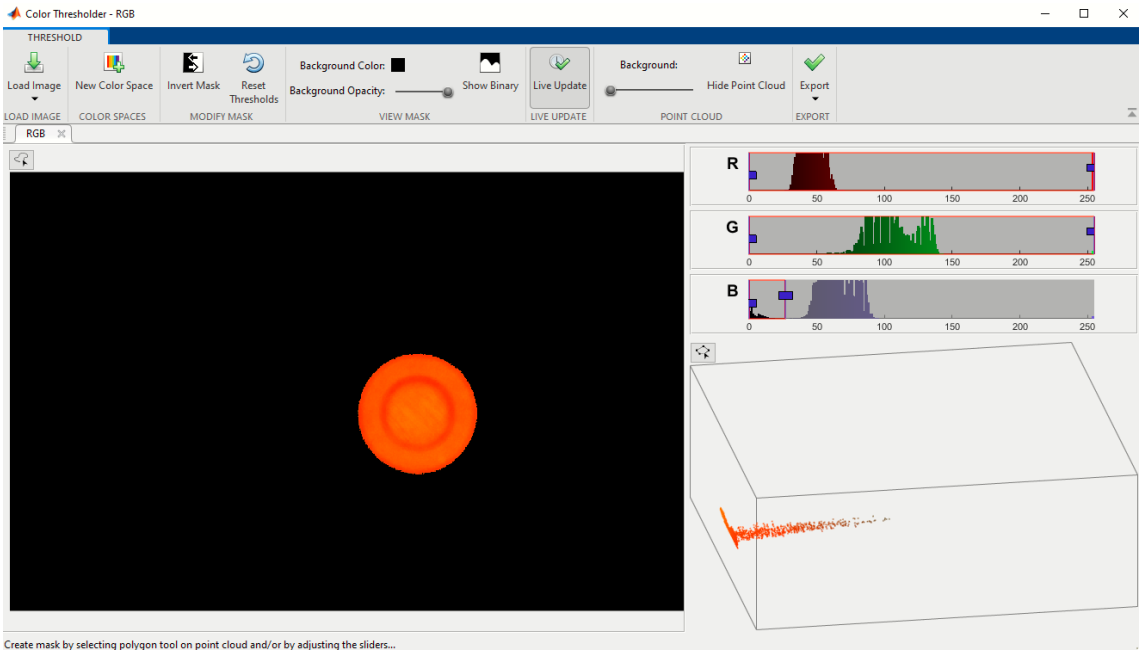


Figure 9: Example of tuning a mask within the Color Thresholder app

8. Export mask
   Once you are satisfied with the mask, you export it by clicking *Export Function* in the dropdown menu under *Export* on the top right. This opens the mask as function in the Editor.

9. Change mask name
   Change the function name in the script, which is set as *createMask* by default.

10. Save mask
    Now save the function in the *Mask* folder, where the default masks are stored as well. The name of the file should match the function name, MATLAB should do this automatically though.

11. Add mask
    Finally you can add the mask to the `camera.m` and `camera_standalone` scripts by replacing one the existing masks. Note that you must not delete the function handle (@).

6 Simulink

The goal of this section is to explain the details of the basic Simulink file.
Simulink is a MATLAB interface that visualizes code in the form of block diagrams. Simulink is used to control the robot arm setup. In order to kickstart the process, a basic Simulink file is given. This functions as the basis for designing the controller.
The highest level in the Simulink file consists of the connection between two subsystems. The *Controller* block contains the basis for the controller. The *RobotArm* block contains parts that are critical for communicating with the setup. Hence **neither** the contents of this block, **nor** the connections between these two subsystems may be changed.

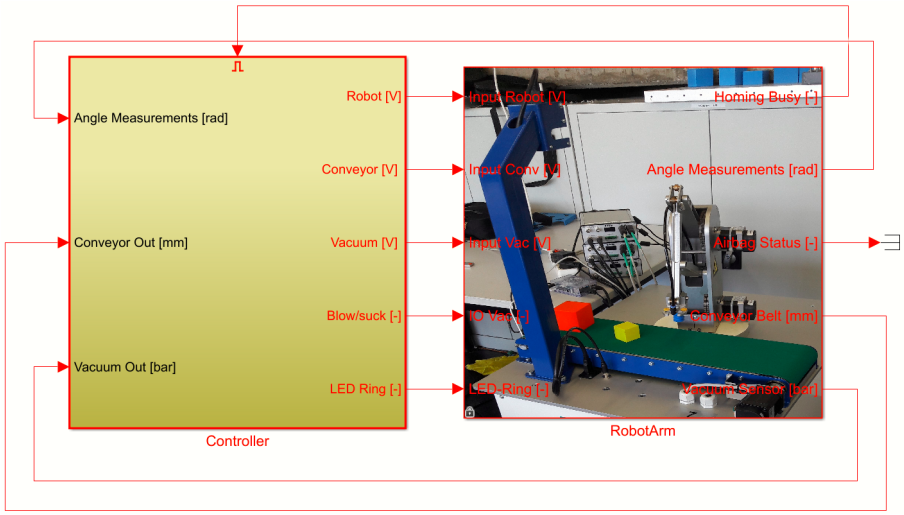


Figure 10: Connection subsystems in Simulink.

Going down one level, in the *Controller* subsystem, you will find the structure below. Essentially all parts within this subsystem may be altered, however it is advised to stick to this general structure. In the remainder of this section all individual parts of this structure will be elaborated upon.

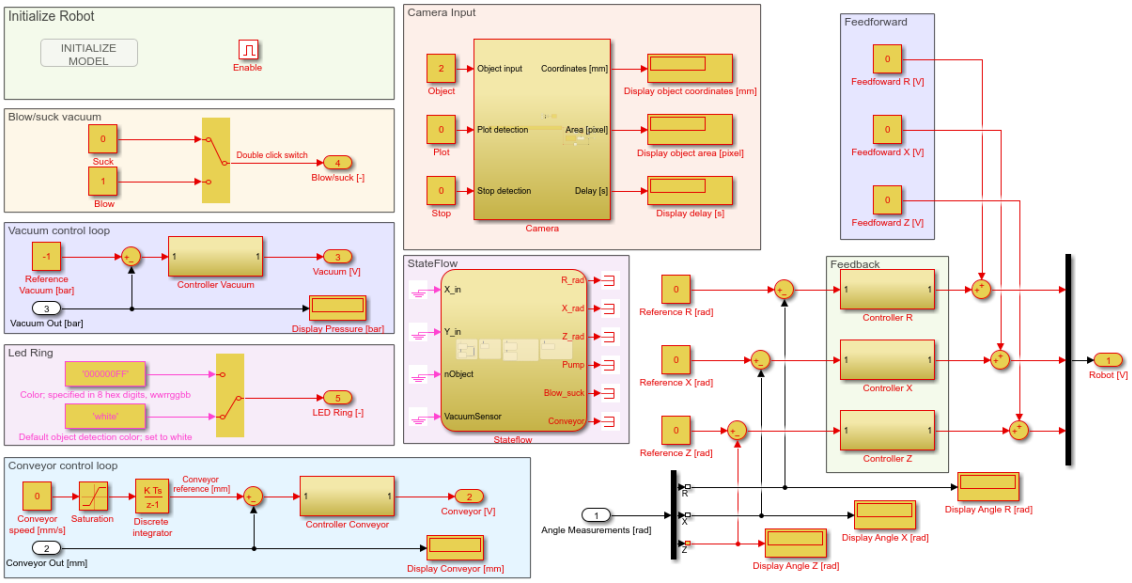


Figure 11: Structure controller subsystem in Simulink.

6.1 Initialize Robot

The *Initialize Model* button loads the calibration values that are specific for every setup. These values contain offsets that make sure that the robot homes to the same position on every setup. On top of this, it compensates for any misalignment of the camera with respect to the robot arm.

The *enable* block tells Simulink when the setup is done homing and that your controller may thus be executed. Hence this block may **not** be touched.

Figure 12: Initialization block.

6.2 Blow/suck vacuum

The end effector of the robot arm has a vacuum gripper attached. This gripper can either suck to grip to an object or blow to release an object. Figure 13 shows the suck/blow block in Simulink. The former is done when the output *Blow/suck* is set to 0, for the latter this output has to be set to 1. In order to do this by hand, a switch is implemented, which toggles between blow and suck by double clicking.

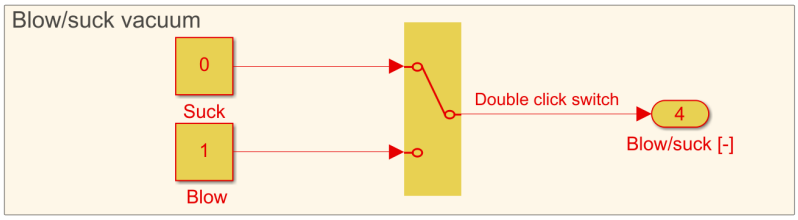


Figure 13: Blow/suck choice.

6.3 Vacuum control loop

Figure 14 shows the vacuum control loop in Simulink. When the vacuum gripper is in suck mode, the strength of the vacuum can be adjusted by changing the value in the *Reference Vacuum* block. The setup can pull a vacuum from 0 bar (atmospheric pressure) to approximately -0.5 bar. By default, the reference is set at -1 bar, which means that the vacuum pump is always running at maximum power. The vacuum is regulated via a simple feedback loop, where the controller is a gain that transforms pressure to voltage. There is no use in changing this controller. The current pressure in the gripper can be seen in the display.

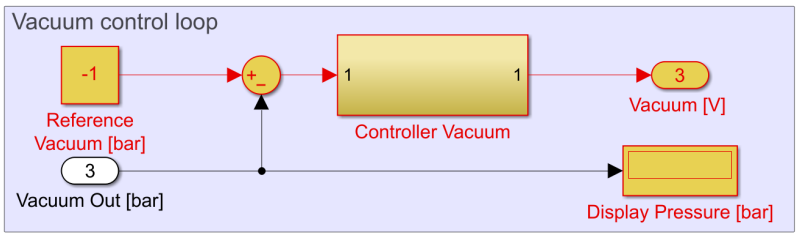


Figure 14: Vacuum control loop.

6.4 LED Ring

The setup also includes a multi-color LED ring. It is located around the camera and can thus illuminate the conveyor belt in different colors. Figure 15 shows the LED ring part in Simulink. Similar to the vacuum gripper you can toggle between two default settings by double clicking the switch. The LED ring understands three different inputs: 'white', 'black' and hex code. Of course, 'white' means white light and 'black' means no light. Alternatively, you can use hex code to select a color of your choosing.

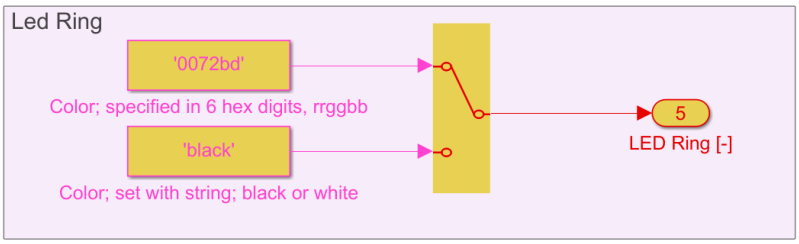
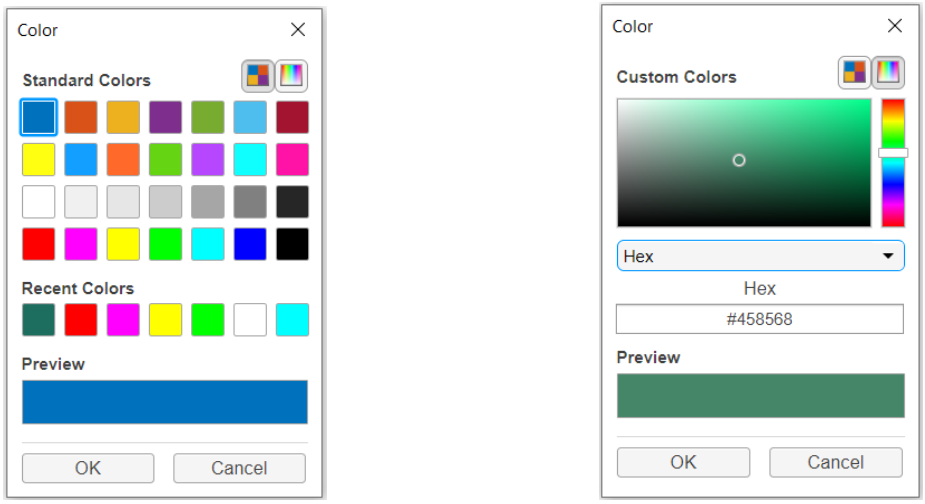


Figure 15: Color selection LED ring.

An easy way to select a color is by using the MATLAB color picker tool. Do this by typing `uisetcolor` in the MATLAB command window. This opens a UI as seen in Figure 16a. In the widget, click the *Custom Colors* tab in the top right. Another windows apprears, as seen in Figure 16b. Then select the color on the palette and also select *Hex* from the dropdown menu. This will generate the hex code of your color. Copy this hex code from the widget to the Simulink block. Do this without the hashtag and within single quotation marks, similar to the default example.



(a) (b)

Figure 16: Color picker tool. (a) Standard colors. (b) Custom colors.

Furthermore, the LED ring indicates the state of the setup. It illuminates orange during homing, flashing green for one second once it has finished. When the robot arm goes out of bounds, the LED ring will be red. The different states are shown in Figure 17.

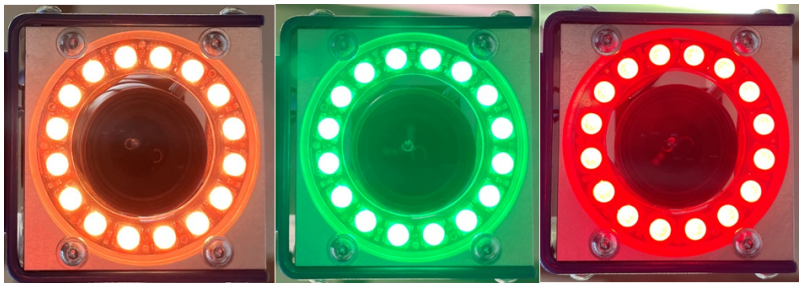


Figure 17: Indication state of setup.

6.5 Conveyor control loop

Figure 18 shows the conveyor control loop in Simulink. In the *Conveyor Speed* block the velocity of the conveyor belt can be changed between -100 mm/s and 100 mm/s. Similar to the vacuum pump, the conveyor belt is controlled by a simple feedback loop. Firstly, the reference velocity is integrated in order to get a reference position. Then the position controller makes sure that the conveyor belt moves smoothly and accurately. Again there is no use in changing this controller. The current conveyor position can be seen in the display.

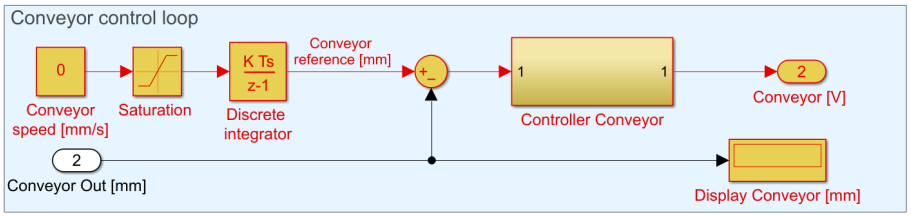


Figure 18: Conveyor control loop.

6.6 Camera Input

When you want to include the camera, for example to locate the objects for the robot arm to pick up, you can let Simulink and the camera communicate via this block. The insides of the *Camera* block transform the actual camera output matrix to separate outputs, so there is no need in changing this block.
It outputs the X and Y coordinates of the objects that the camera detects. It also outputs the area of every object. The camera can detect at most three objects. Next to this, it outputs the delay of the camera. That is the time it takes for the signal to travel from the camera to Simulink. Because of this delay, the objects are actually at a slightly different location than Simulink expects. This can become problematic for a fast moving conveyor belt, hence can be accounted for with this output.
The input *Object* determines type of objects that the camera detects, meaning objects within a specific color and area range. The camera can detect three different object types (inputs 1, 2 & 3). What these types are and how to change them can be read in section 5. The input *Plot* can be used to toggle the live video feed of the camera on (input 1) and off (input 0). Lastly, the input *Stop* is used to stop running the camera (input 1). This is the same as stopping the `camera.m` script directly.

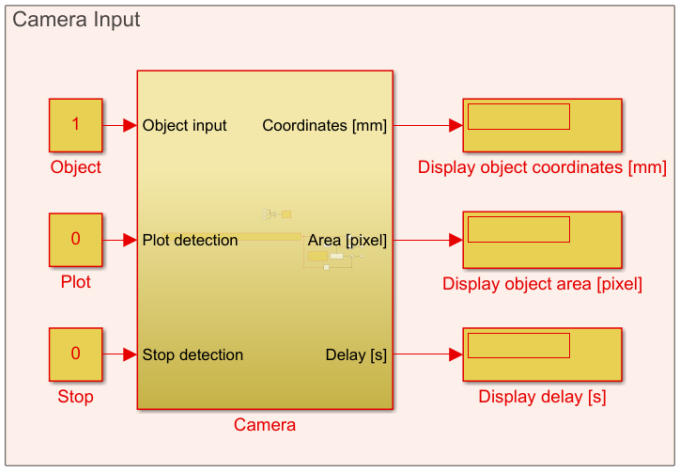


Figure 19: Camera block.

6.7 Stateflow

Stateflow is a tool within Simulink to make flowcharts that can be used to control the state of your system. On the MATLAB website there is plenty of information to get to know this tool. You can also directly access the interactive beginners guide from the MATLAB command window by typing: `learning.simulink.launchOnramp('stateflow')`.
The default inputs and outputs from Stateflow are connected to grounds and terminators respectively, in order to prevent Simulink from giving errors for unconnected signals. These can be deleted once you connect the inputs and outputs to other blocks.

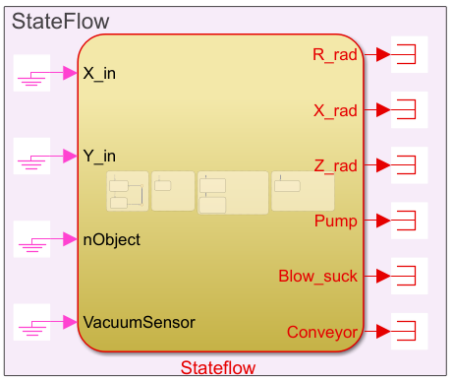


Figure 20: Stateflow block with inputs and outputs.

Inside the Stateflow subsystem there is already an example Stateflow chart that can be used as a basis. This Stateflow example is shown in Figure 21 It includes some basic states for different parts of the setup, with some basic connections, conditions and events.

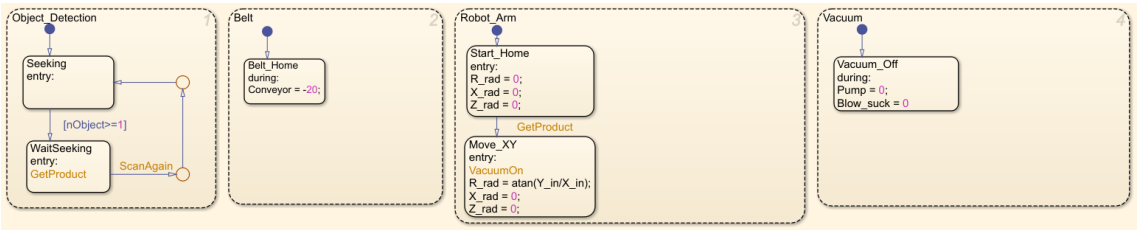


Figure 21: Example Stateflow.

6.8 Controller

The controller part of the Simulink file is where you can implement your controllers. It has a feedback loop already build into it, which sends the control signal to the plant (robot arm setup) and receives the measurements to close the loop. In the *Feedback* part there are three subsystems for the controller of each motor. Similarly, there are feedforward and reference inputs per motor.

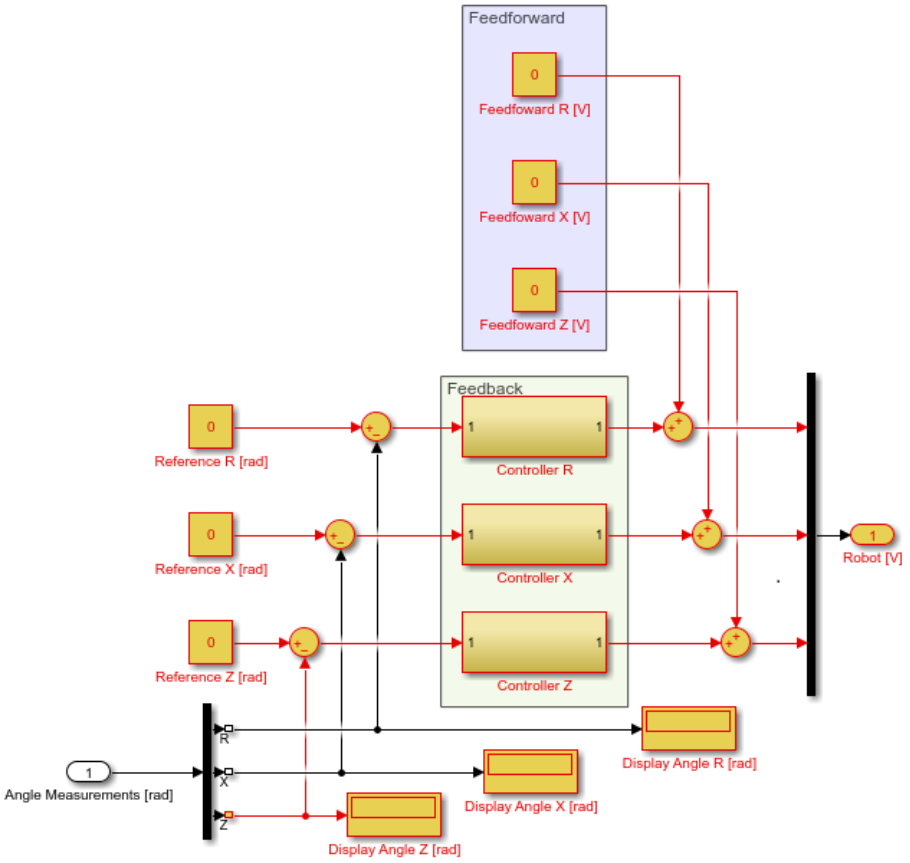


Figure 22: Controller robot-arm.

6.9 Trajectory generation

Two custom blocks have been created in order to generate a smooth trajectory while the system is running. These blocks are called the quintic polynominal trajectory blocks as they generate a trajectory based on a quintic polynomoinal. These blocks can be found under the tab *Simulation* in *Library Browser* and then *Robotarm library*.

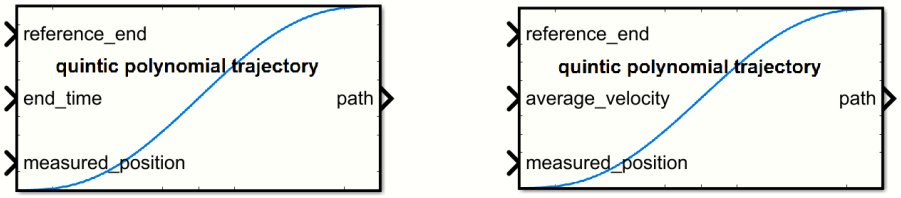


Figure 23: The quintic polynominal trajectory blocks

One of the blocks can be controlled using the desired end time and one block can be controlled using the desired average velocity. The inputs of the blocks are the end reference, end time or average velocity and the measured position. The end reference is the position in which you want the trajectory to end. This value should only be changed if the desired position is reached. Do not use a variable that changes it's value at every time-step. This will result in a new generated path at every time-step and a smooth trajectory will not be achieved. The end time is the time in seconds it takes to go to the end position. This is useful for when you want the robot at a position at a certain time. The average velocity is the average velocity that the generated reference will have. This is useful for when you want the robot to have a certain speed, no matter the distance it has to travel (unlike when the end time is used). The measured position is the current measured position of the robot-arm. The encoder values in radians of the R, X or Z value should connect to this input. The output is a generated path. When the path has reached the end reference, the end reference can be changed to another value and the path will go smoothly to the new end reference. Figure 24 shows an example trajectory with reference_end = 1 and end_time = 2 as input.

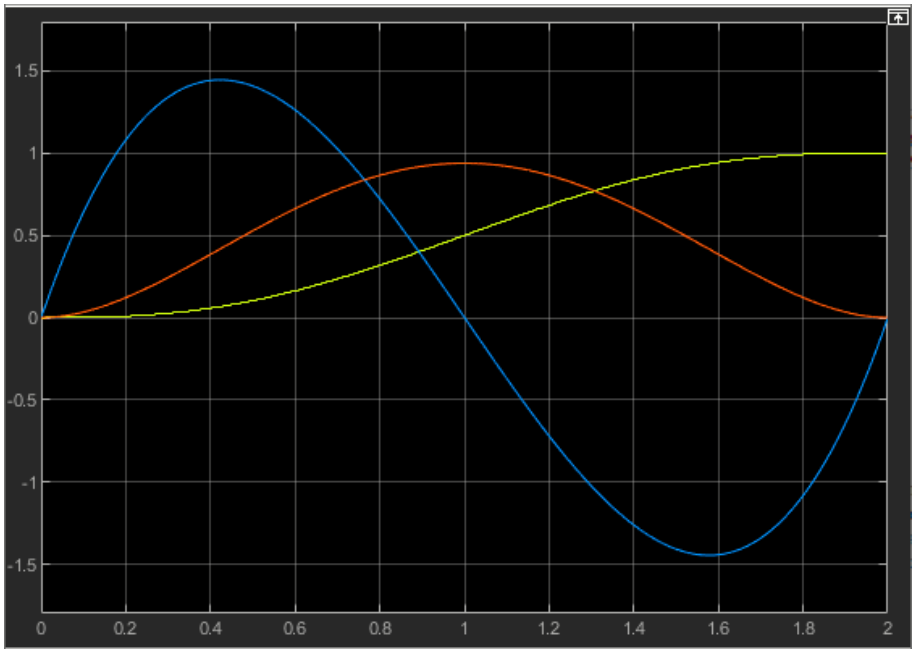


Figure 24: Example trajectory with reference_end = 1 and end_time = 2 as input. The yellow line shows the position, the red line the velocity and the blue line is the acceleration.

6.10 Measurement data

Measurement data is the data that is generated by the setup or Simulink. Because the model runs on the Raspberry Pi, the data is initially stored there as well.

6.10.1 Realtime scope

This block is a variant on the regular Simulink Scope block. It can be used to view signals live, thus while the Simulink model is running. Moreover, it does not save the data. The difference with the regular scope is that Realtime scope runs at a lower sample frequency of 500Hz. Therefore it is computationally less demanding. To use this block, click anywhere in Simulink and type `Realtime Scope`.

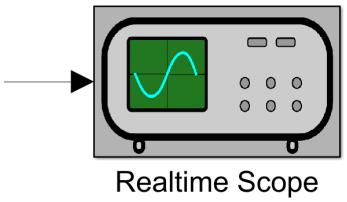


Figure 25: The Realtime Scope block.

6.10.2 To file

A general option to record data is via the To File block. Add the block by clicking anywhere in Simulink and typing `To File`. Connect it to the signal you want to record and open the block by double-clicking it. In here choose a file name and variable name. Also use the save format *Array* instead of *Timeseries*. If you want to record multiple signals in a single file, place a `Mux` block in front of the `To File` block.
This saves the data on the Raspberry Pi. The data can be retrieved by executing:
`SPERTE_Get_Latest_Measurement(p,'mat',N);` in the command window. In this function:

- `p` is the `raspi()` object made earlier.
- `'mat'` is the file type
- `N` is the number of files you want to retrieve, in other words, the number of `To File` blocks.

The files are then stored in a folder *measurements* in your current directory. Note that this can only be done once you have stopped the Simulink model from running.

Figure 26: The To File block.

6.10.3 Measurement block

The Measurement Block does about the same as the `To File` block. Add the block by clicking anywhere in Simulink and typing `Measurement Block`. Connect it to the signal you want to record and open the block by double-clicking it. In here, fill in the number of samples you want it to record. If you want to record multiple signals in a single file, place a `Mux` block in front of the `To File` block.
This saves the data on the Raspberry Pi. The data can be retrieved by executing:
`data = SPERTE_Measure_And_Collect(p,N_signals,N_samples,ModelName,ShouldPlot);` in the command window. In this function:

- `p` is the `raspi()` object made earlier.
- `N_signals` is the number of signals.
- `N_samples` is the number of samples.
- `ModelName` is the name of the current model.
- `ShouldPlot` is a flag to enable plotting in a figure, with 1 is plotting and 0 is no plotting.

This function returns the measured data as the variable *data* in your workspace.

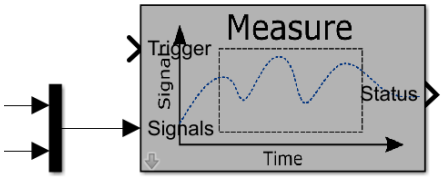


Figure 27: The Measurement Block.

6.10.4 RTScope and FRF measurement

RTScope is a useful tool to view signals in real-time. RTScope also has the option to measure a Frequency Response Function (FRF) in real-time. This subsection explains how to measure a FRF of one of the axis of the robot arm with RTScope. A separate Simulink file is created with the RTScope block, named `robotarm_student_2021a_RTScope_v2.slx`. Figure 28 shows the seperate Simulink file. This is the original Simulink model, but expanded with a RTScope block and a signal generator block called Ref3.

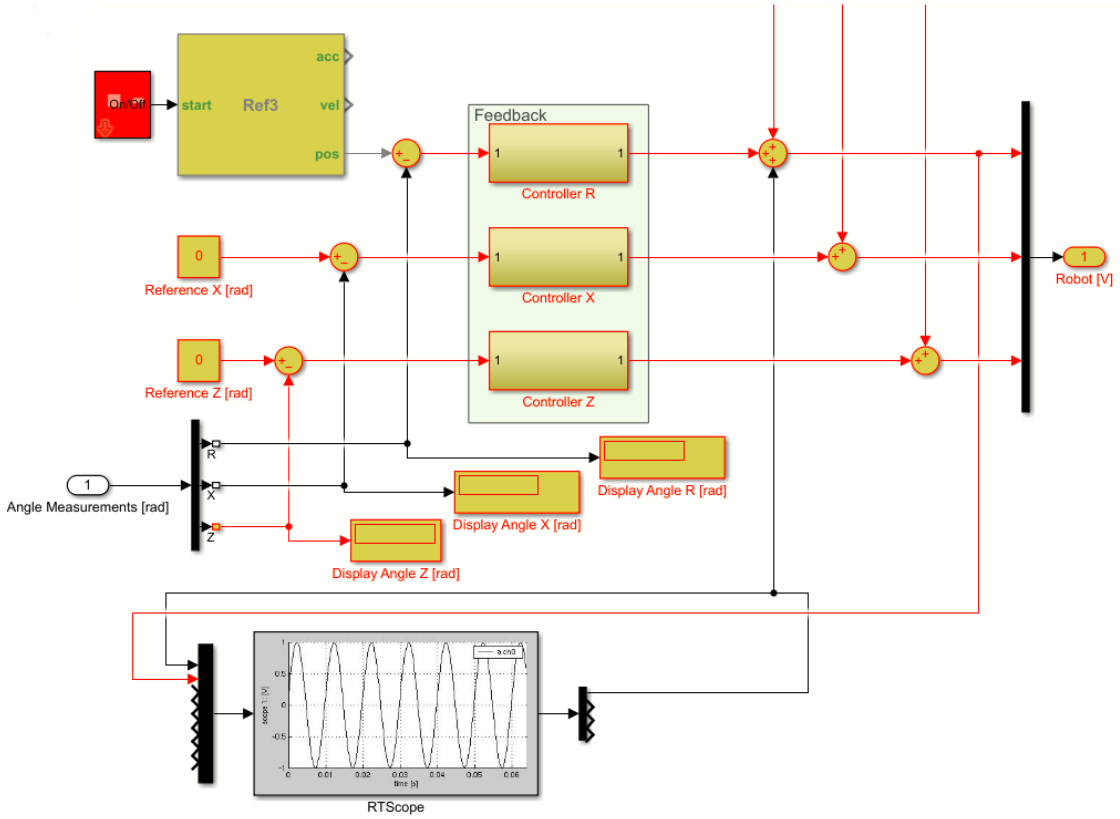


Figure 28: Adapted Simulink file to measure the FRF of the R-axis of the robot arm.

The Ref3 block implements a smooth trajectory for the system to follow that can be used to improve the quality of the FRF measurements. In order to use this signal generator, open the Ref3 block and change the parameters to create your own smooth trajectory. After the desired trajectory is created, *Accept* the trajectory. Accept the trajectory every time you open the Simulink model. Set the ref power (red block) to *on* and the reference will be generated. Do these steps before running the model.

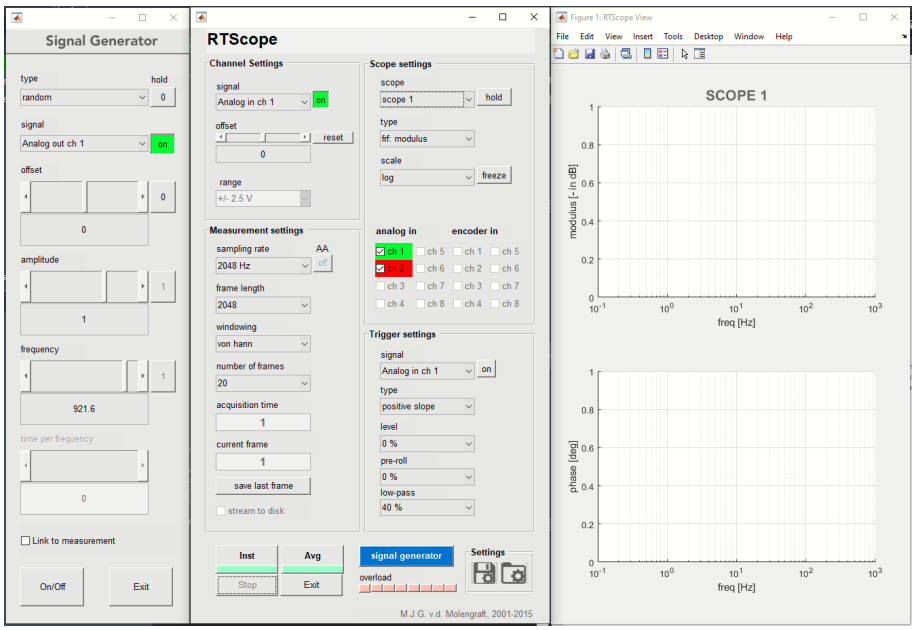


Figure 29: GUI of the RTScope block.

The RTScope block is connected to several signals to measure the sensitivity function of the R-axis. The block can be opened by double clicking on the block while the model is running.

Figure 29 shows the GUI of RTScope. To measure the sensitivity function of the system set the following RTScope settings:

- Channel settings
  - signal: Analog out ch 1: ON
  - signal: Analog out ch 2: ON
  - offset: 0
- Measurements settings
  - sampling rate: 2048 Hz
  - frame length: 2048
  - windowing: von hann
  - number of frames: 20
  - offset: 0
- Scope settings for sensitivity
  - scope: scope 1
    - type: frf: modulus
    - scale: log
  - scope: scope 2
    - type: frf: phase
    - scale: log
- Signal generator
  - type: random
  - signal: Analog out ch 1: ON
  - offset: 0
  - amplitude: 0.5
  - frequency: 921.6

After these settings are set, press the *On/Off* button on the signal generator tab to generate a noise signal. To measure the sensitivity function press *Avg* in the bottom left. To measure the open loop instead of the sensitivity function, change the type of scope 1 and 2 to sens2ol: modulus and sens2ol: phase, respectively. When the FRF looks sufficient, the data can be saved by saving the figure: Figure 1: RTScope view as .fig file. This can be done by going to *file* and then *save as*. Use the function `RTScope_figure2FRF_data` to convert the figure to FRF data that is compatible with *shapeit*. This function can be called from the command window. Example command: `>>[FRF, hz] = RTScope_figure2FRF_data('figure_name');`. When the measurements are finished, please close RTScope before stopping the model to avoid MATLAB from freezing/crashing.