INTERNSHIP REPORT

The report is submitted as part of requiremnt of Master program Robotic System Engineering in RWTH Aachen University. The internship was conducted in BMW, Leipizig in Logistics Robotics Department.

By:
Anshul Pagariya
Student: Msc Robotic System Engineering
Matr. No. : 404634

Supervisior:
Mr. Vadim Paul
Employee BMW group

DECLARATION

I, Anshul Pagariya, hereby affirm that the present work was written independently, and no other sources and tools were used than those indicated. All pieces of work that have been taken literally or in essence from other sources, as such are marked.

Leipzig, DATE

Anshul Pagariya

TABLE OF CONTENT

ABOUT BMW GROUP

Bayerische Motoren Werke AG, commonly referred to as BMW, is a German multinational corporate manufacturer of luxury vehicles and motorcycles headquartered in Munich, Bavaria, Germany. The corporation was founded in 1916 as a manufacturer of aircraft engines, which it produced from 1917 until 1918 and again from 1933 to 1945.

Automobiles are marketed under the brands BMW, Mini and Rolls-Royce, and motorcycles are marketed under the brand BMW Motorrad. In 2017, BMW was the world's fourteenth-largest producer of motor vehicles, with 2,279,503 vehicles produced. The company has significant motorsport history, especially in touring cars, Formula 1, sports cars and the Isle of Man TT.

BMW is headquartered in Munich and produces motor vehicles in Germany, Brazil, China, India, Mexico, the Netherlands, South Africa, the United Kingdom, and the United States. The Quandt family is a long-term shareholder of the company (with the remaining shares owned by public float), following brothers Herbert and Harald Quandt's investments in 1959 which saved the company from bankruptcy.

Bayerische Flugzeugwerke AG was formed in 1916. This company was renamed to Bayerische Motoren Werke (BMW) in 1922. However the name BMW dates back to 1913, when the original company to use the name was founded by Karl Rapp (initially as Rapp Motorenwerke GmbH). BMW's first product was a straight-six aircraft engine called the BMW IIIa, designed in the spring of 1917 by engineer Max Friz. Following the end of World War I, BMW remained in business by producing motorcycle engines, farm equipment, household items and railway brakes. The company produced its first motorcycle, the BMW R 32 in 1923. BMW became an automobile manufacturer in 1928 when it purchased Fahrzeugfabrik Eisenach, which, at the time, built Austin Sevens under licence under the Dixi marque. The first car sold as a BMW was a rebadged Dixi called the BMW 3/15, following BMW's acquisition of the car manufacturer Automobilwerk Eisenach. Throughout the 1930s, BMW expanded its range into sports cars and larger luxury cars.

Aircraft engines, motorcycles, and automobiles would be BMW's main products until World War II. During the war, against the wishes of its director Franz Josef Popp,[citation needed] BMW concentrated on aircraft engine production using forced labor consisting primarily of prisoners from concentration camps, with motorcycles as a side line and automobile manufacture ceased altogether. BMW's factories were heavily bombed during the war and its remaining West German facilities were banned from producing motor vehicles or aircraft after the war. Again, the company survived by making pots, pans, and bicycles. In 1948, BMW restarted motorcycle production. BMW resumed car production in Bavaria in 1952 with the BMW 501 luxury saloon. The range of cars was expanded in 1955, through the production of the cheaper Isetta microcar under licence. Slow sales of luxury cars and small profit margins

from microcars meant BMW was in serious financial trouble and in 1959 the company was nearly taken over by rival Daimler-Benz. A large investment in BMW by Herbert Quandt and Harald Quandt resulted in the company surviving as a separate entity. The BMW 700 was successful and assisted in the company's recovery.

The 1962 introduction of the BMW New Class compact sedans was the beginning of BMW's reputation as a leading manufacturer of sport-oriented cars. Throughout the 1960s, BMW expanded its range by adding coupe and luxury sedan models. The BMW 5 Series mid-size sedan range was introduced in 1972, followed by the BMW 3 Series compact sedans in 1975, the BMW 6 Series luxury coupes in 1976 and the BMW 7 Series large luxury sedans in 1978.

The BMW M division released its first road car, a mid-engine supercar, in 1978. This was followed by the BMW M5 in 1984 and the BMW M3 in 1986. Also in 1986, BMW introduced its first V12 engine in the 750i luxury sedan.

The company purchased the Rover Group in 1994, however the takeover was not successful and was causing BMW large financial losses. In 2000, BMW sold off most of the Rover brands, retaining only the Mini brand.

In 1998, BMW also acquired the rights to the Rolls Royce brand from Vickers Plc.

The 1995 BMW Z3 expanded the line-up to include a mass-production two-seat roadster and the 1999 BMW X5 was the company's entry into the SUV market.

The first modern mass-produced turbocharged petrol engine was introduced in 2006, (from 1973 to 1975, BMW built 1672 units of a turbocharged M10 engine for the BMW 2002 turbo), with most engines switching over to turbocharging over the 2010s. The first hybrid BMW was the 2010 BMW ActiveHybrid 7, and BMW's first mass-production electric car was the BMW i3 city car, which was released in 2013, (from 1968 to 1972, BMW built two battery-electric BMW 1602 Elektro saloons for the 1972 Olympic Games). After many years of establishing a reputation for sporting rear-wheel drive cars, BMW's first front-wheel drive car was the 2014 BMW 2 Series Active Tourer multi-purpose vehicle (MPV).

In January 2021, BMW announced that its sales in 2020 fell by 8.4% due to the impact of the COVID-19 pandemic and the restrictions. However, in the fourth quarter of 2020, BMW witnessed a rise of 3.2% of its customers' demands.

INTRODUCTION

LOGISTICS ROBOTICS:

In the department of Logistics Robotics of BMW focus is on developing Robots for Logistics applications like Depallatizing, Palletizing, Picking, and Placing Boxes and other Items in BMW's Logistics area. I had the opportunity to work on two of such Robots, Sortbot and Splitbot. In the following paragraph brief explanation about each of them is given.

1. SORTBOT:

BMW Group has developed SORTBOT for the application of palletizing the incoming empty boxes (KLT) from the plant. Universal robot's UR10 is used in the Sortbot. The palletizing process is shown in figure 2 below.

Figure 1: Introduction to Sortbot.

Figure 2: Different postion's of Sortbot

At Location 1, marked as 1 in figure 2, the robot with the use of a camera and Artificial intelligence detects the KLT and picks it.
At Location 2, marked as 2 in figure 2, the robot flips the KLT and grips it from the turned side.
At Location 3, marked as 3 in figure 2, the robot palletizes the KLT on the wooden pallet.

2. SPLITBOT:

Another Robot developed by BMW for Logistics application is called Split bot. The purpose of this robot is to depalletize the incoming pallet with KLTs and split them according to empty or nonempty KLT and finally put the KLT on the corresponding Conveyer belt. Each empty and filled KLT will go on different Conveyer belts. The camera on the gripper with the help of Artificial intelligence detects the location and status (empty or non-empty) of the KLT.

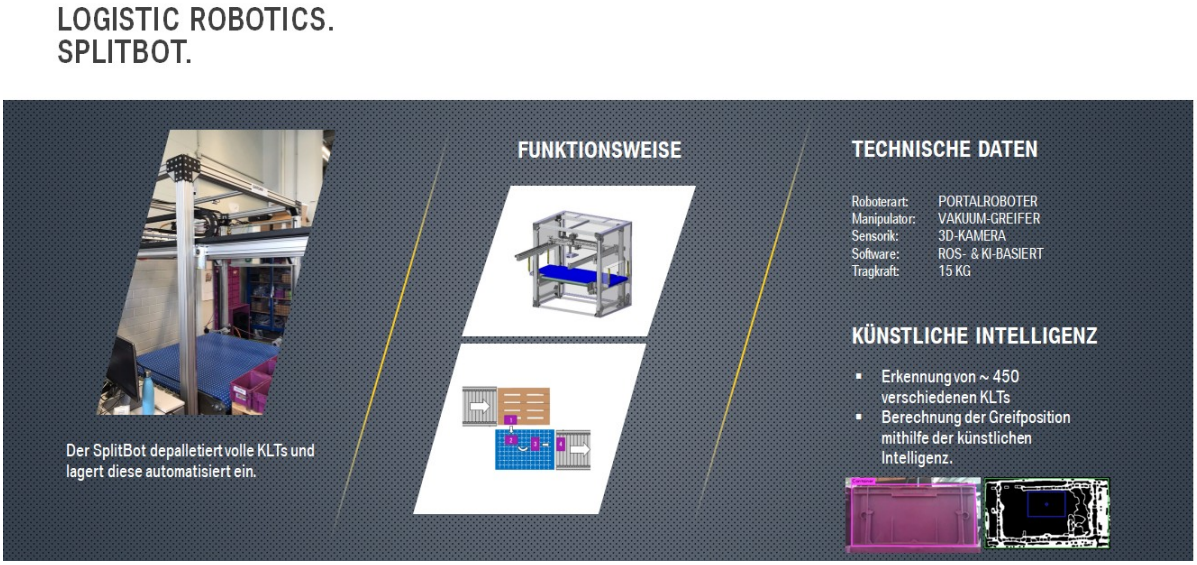


Figure 3: Intoduction to Splitbot

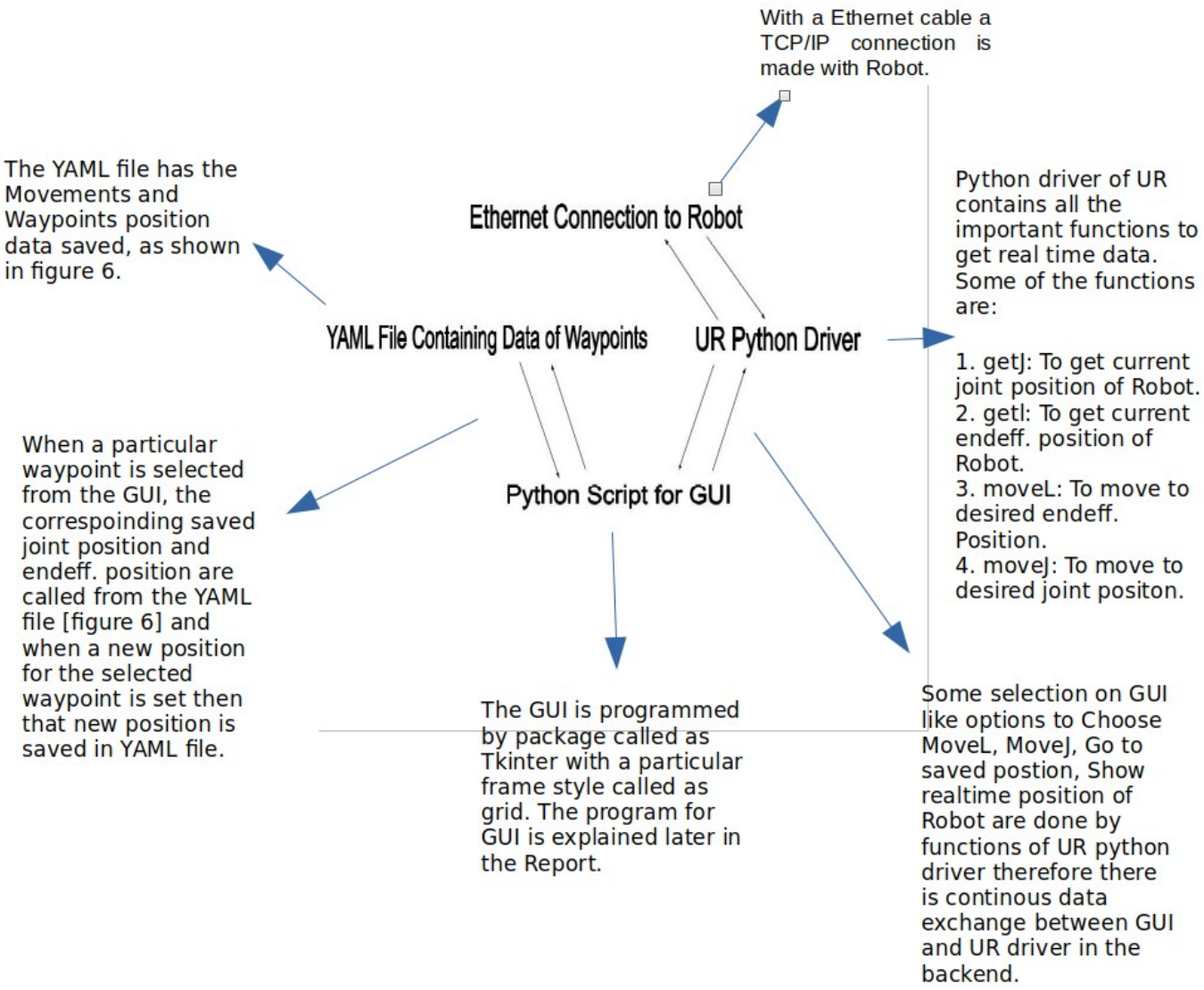
Figure 4: Full view of Splitbot

Figure 5: Gripper approaching KLT

TASKS RELATED TO SORT BOT

1. CREATING GUI TO TEACH WAYPOINTS FOR SORT BOT:

As described on page 6 about the working of Sortbot. To enable Sortbot to do the job of palletizing several points need to be set for the path of the end effector of the Robot, particularly at location 2 [refer to Figure 2], because here the KLT would be flipped around by the robot. To make it easy for anyone to set these points I was tasked to create a GUI. The GUI was made using a software package called Tkinter which is programmed in Python. The GUI has dynamic features so that It can respond to any selection made on the GUI. The chart below shows the integration done in GUI.



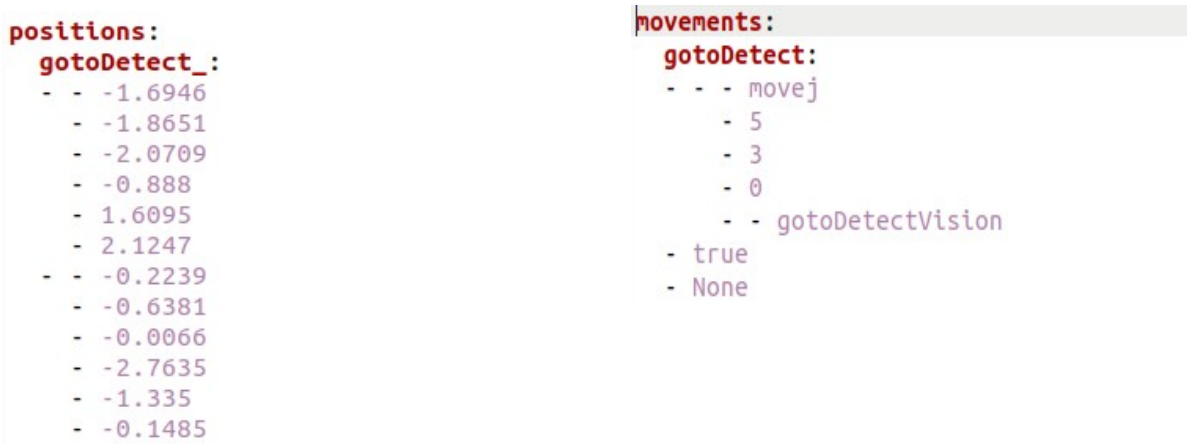


Figure 6: Positions [In left] and Movements [in Right] data shown.

Figure 6, Shows waypoint joint position and end eff. position [In left] and movements collection of waypoints with velocity, acceleration and blend radius details [in Right].

2. Picture of GUI:

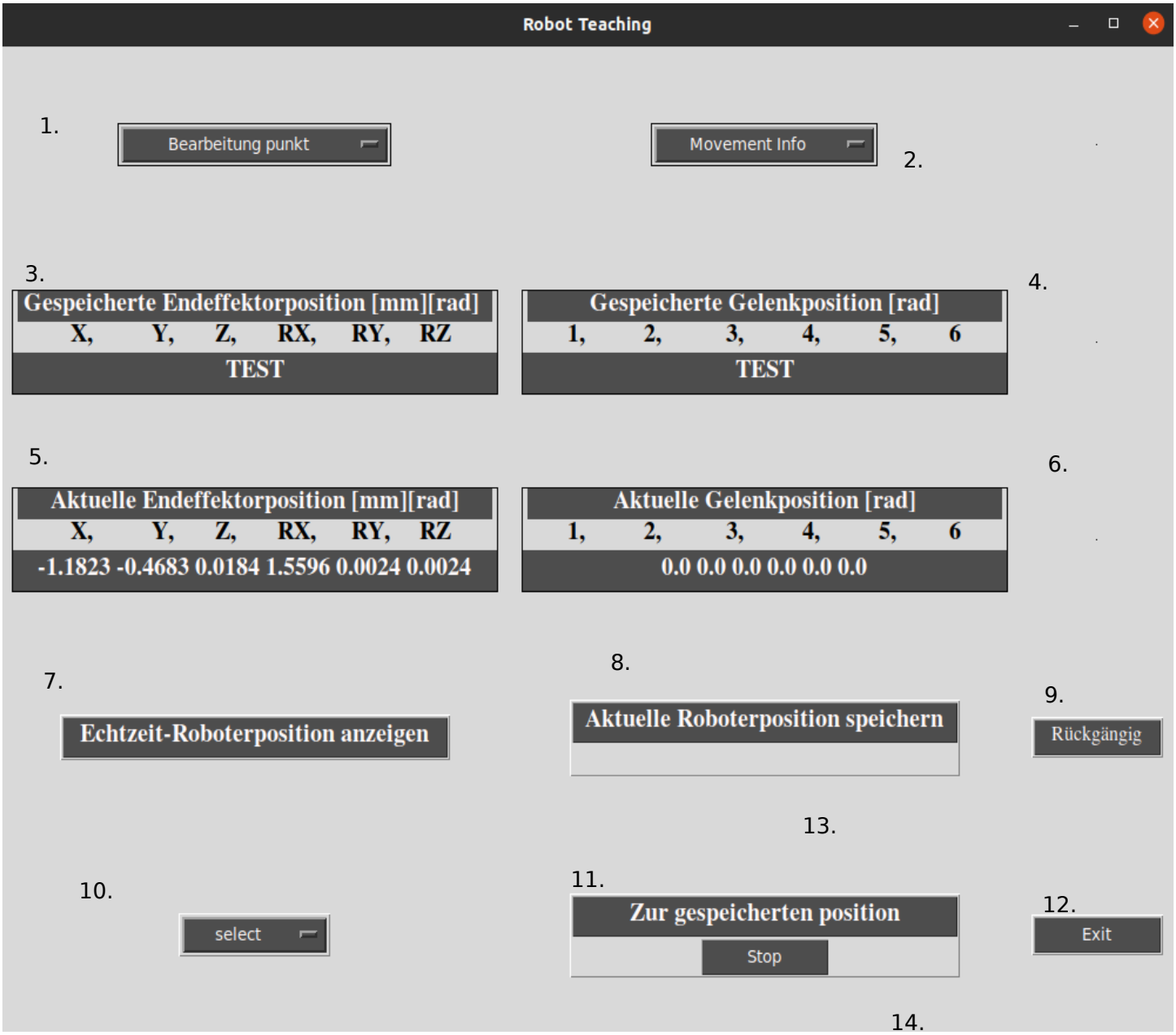


Figure 7: GUI to teach Sortbot

1. Is a drop-down menu button and can be used to select the waypoint of interest, figure 8 shows the expanded list.

2. A Few waypoints are defined together as Movement (refer to figure 6) with information about velocity, acceleration, and blending radius, this drop-down

menu button gives information about them, figure 10 and figure 11 show these functions.

3. This block shows the saved endeff. position for selected waypoint.

4. This block shows the saved joint position for the selected waypoint.

5. This block shows the current Robot endeff. position.

6. This block shows the current Robot joint position.

7. This button can be used to update the real-time Robot positions to be displayed on GUI.

8. This button can be used to the save current Robot position for the selected waypoint.

9. This button can be used to undo the change made through GUI.

10. This button can be used to select MoveJ or MoveL for the movement.

11. This button can be used to go to the saved position of the waypoint.

12. This button is used to exit GUI.

13. This button notifies the changes or actions done through GUI.

14. This button can be used to stop the Robot during movement.

Figure 8: Drop down menu to select a waypoint

Figure 9: Drop down menu to select movej or movel

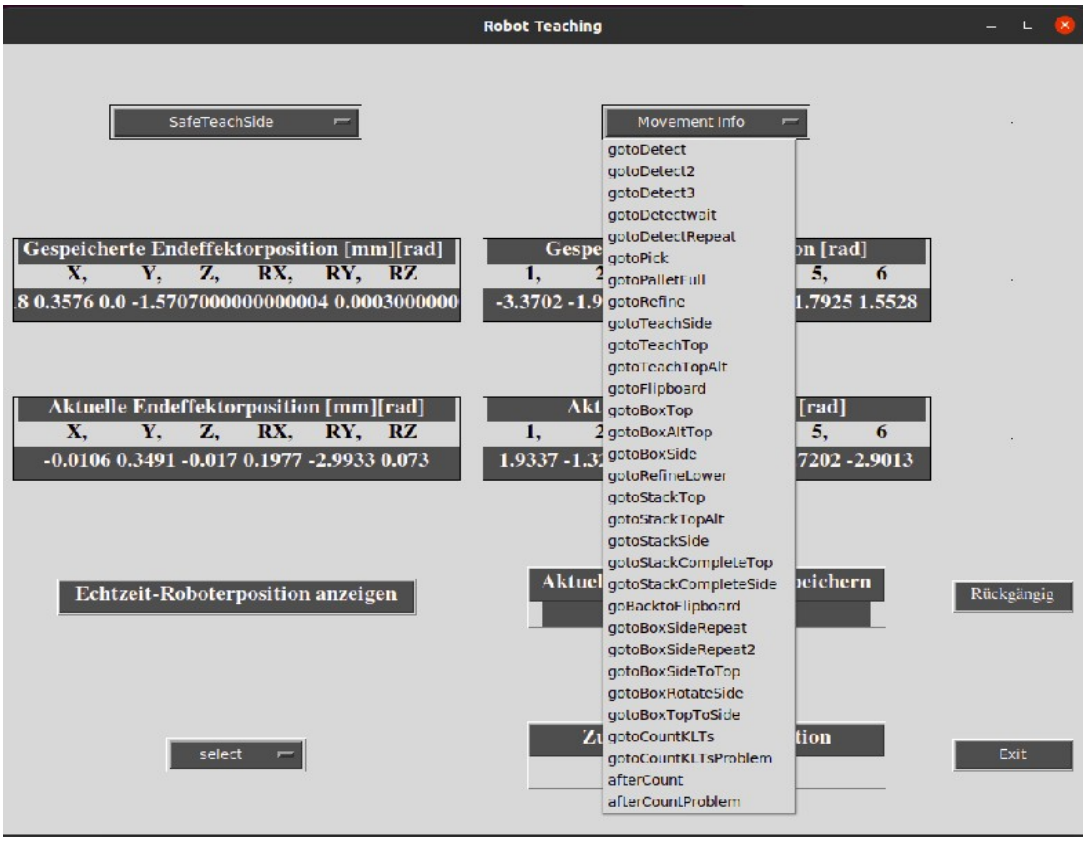


Figure 10: Drop down menu to select Movement info

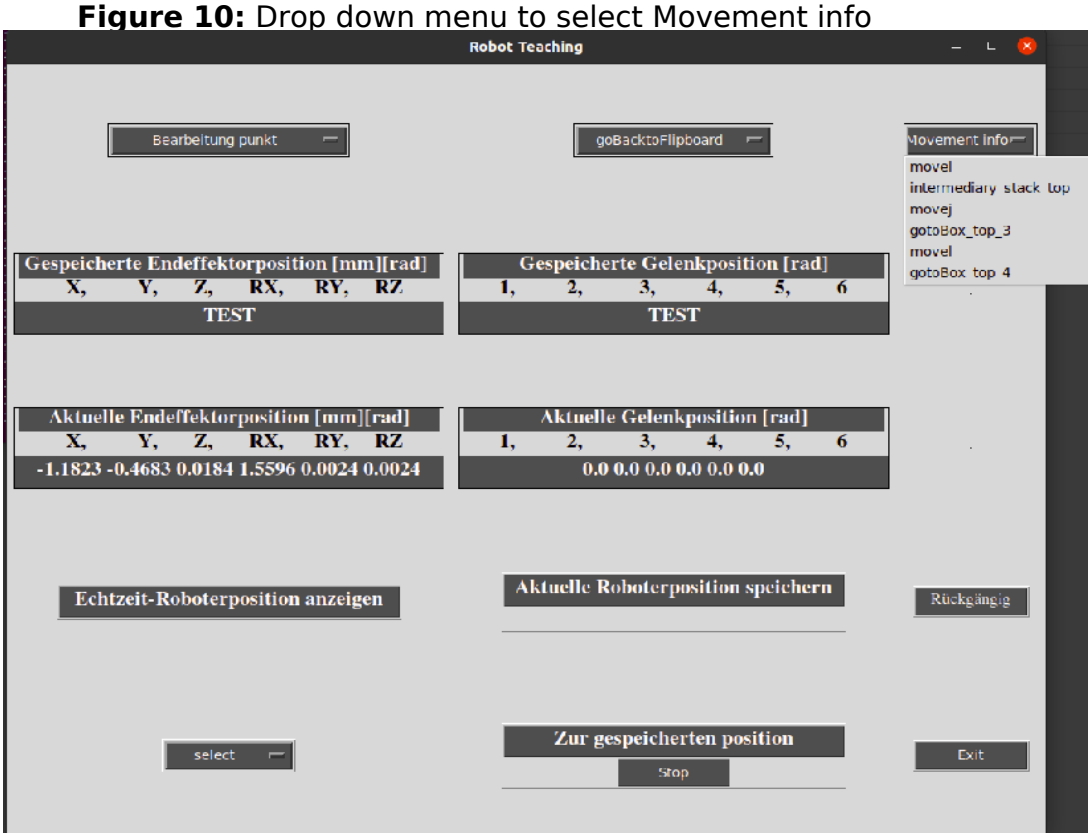


Figure 11: Expanded list of Information for selected movement

3. Explanation of code to generate the GUI:

The script was made using Tkinter [SOURCE] and python. In figure 12, Initially, the acceleration and velocity of the robot are set. After which a TCP\IP connection to Robot is made. Then the tk window object is created. Then different files path for different Sortbot versions were specified. Then the name of different tabs in GUI was given.

```
a = 0.5
v = 0.5
rob = urrobot.URRobot('192.168.0.3', enable_rtde_input=True)
window = tk.Tk()
window.geometry("1000x800")  # Size of the window
window.title("Robot Teaching")  # Adding a title

### Text parameter

right_robot_param_file = '/home/robotics/Desktop/software-sortbot/sortbot_ws/src/sortbot_state_machine/include/sortbot_7/sortbot_motion_right_3147.yaml'

left_robot_param_file = '/home/robotics/Desktop/software-sortbot/sortbot_ws/src/sortbot_state_machine/include/sortbot_7/sortbot_motion_left_3147.yaml'

inno_garage_file = '/home/q521154/Desktop/Desktop/sortbot_motion_left_4147.yaml'

#Mention the yaml file according to robot

yaml_file = left_robot_param_file



text_update_position = 'Gespeicherte position aktualisiert' #Gespeicherte position aktualisiert  #'Saved Position Updated'

text_to_guide_to_select_one_option_in_drop_down_menu = 'Bearbeitung punkt'              #Wählen Sir eine   #'Select One'

text_to_go_to_saved_position= 'Zur gespeicherten position'  #Zur gespeicherten position  #'Go to saved position'

text_to_save_current_robot_position = 'Aktuelle Roboterposition speichern' #Aktuelle Roboterposition speichern #"Save current robot position"

text_to_show_saved_position =  'Gespeicherte Gelenkposition [rad]' #Gespeicherte Gelenkposition  #"Saved Joint Position"

text_to_show_saved_end_effector_position =  'Gespeicherte Endeffektorposition [mm][rad] ' #Gespeicherte Endeffektorposition  #"Saved End Effector Position"

text_to_show_updated_real_time_position =  'Echtzeit-Roboterposition anzeigen' #Echtzeit-Roboterposition anzeigen  #"Show Robot's real time position"

text_to_show_current_robot_joint_position =  'Aktuelle Gelenkposition [rad]' #Aktuelle Gelenkposition  #"Current joint Position"

text_to_show_current_robot_endeffector_position=  'Aktuelle Endeffektorposition [mm][rad]' #Aktuelle Endeffektorposition  #"Current End Effector Position"

text_to_undo = 'RÜckgängig'

text_to_undo_text = 'RÜckgängig Erfolgreich'
```

Figure 12: Parameter setting in the script

The code shown below in figure 13 is for rows 3, 4, and column 0, 1, 2 of the GUI. In the script, the window (GUI) is divided into several frames. Here Frame_4 is shown in which code for several functionally is added. In row 3 and column 0 button to update the data of real-time robot position is created, the command parameter has the corresponding function which gets called when this button is pressed. Similarly in row 3 and column 1, the function for undo is defined. In row 4 and column 1 a drop-down menu to select movej and movel function is created. When a selection from a drop-down menu is done the function corresponding to that drop-down menu is called for eg. when here any of move j or move l is selected function my_show_ (shown in figure 15) is called through the tracing code, which is shown in figure 14.

```
for i in range(3,5):
    for j in range(3):
        window.columnconfigure(j, weight=1, minsize=100)
        window.rowconfigure(i, weight=1, minsize=100)
        frame_4 = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame_4.grid(row=i, column=j, padx=5, pady=5)
        if i == 3 and j == 0:
            update_current_robot_pose_data()
            button_2 = tk.Button(frame_4, text=text_to_show_updated_real_time_position, width=30, height=1, bg="gray30", fg="snow", command=update_current_robot_pose_data, font=("Times", "15",
"bold") )
            button_2.grid()

        if i == 3 and j == 2:
            button_4 = tk.Button(frame_4, text=text_to_undo, width=10,  bg="gray30", fg="snow", command=undo, font=("Times", "12", "normal") )
            button_4.grid()

        if i == 4 and j == 0:
            my_list = ['movej', 'movel']
            options1 = tk.StringVar(frame_1)
            options1.set('select') # default value

            om2 =tk.OptionMenu(frame_4, options1, *my_list)
            om2.grid()
            om2.config(bg="gray30", fg="snow", width = 10)
            str_out1=tk.StringVar(frame_4)
            str_out1.set("Output4")

        if i == 4 and j == 1:
            button_2 = tk.Button(frame_4, text=text_to_go_to_saved_position, width=30, height=1, bg="gray30", fg="snow", command=go_to_saved_position, font=("Times", "15", "bold") )
            button_2.grid()

        if i == 3 and j == 1:
            button_3 = tk.Button(frame_4, text=text_to_save_current_robot_position, width=30, height=1,bg="gray30", fg="snow", command=replace_and_save_current_position , font=("Times", "15",
"bold")  )
            button_3.grid()

            label_3_2_8 = tk.Label(frame_4, text="", width=30 ,  font=("Times", "15", "bold")  )
            label_3_2_8.grid()
```

Figure 13: Code to set up to creates features in the frame like Buttons, Drop down menu.

```
options1.trace('w',my_show_)
```

Figure 14: Tracing function my_show_ when drop down menu for movej or movel is pressed.

Similarly when a waypoint is selected from the drop-down menu correspondingly my_show (refer to figure 15) is used to show the join positions and end effector saved for the selected point. When the drop-down menu for movements is selected then my_show2 (refer to figure 15) is called, and in this function, the yaml file is opened, and then selected movement is opened and that movement is converted into the list and another drop-down is spawned with that information.

```
    def my_show(*args): ##function called when options value changes
        str_out.set(options.get())
        _, _, current_point_data= yaml_data()
        label_3_1_6.config(text = current_point_data[options.get()][0])
        label_3_1_3.config(text = current_point_data[options.get()][1])
        label_3_2_8.config(text = '',bg="gray30", fg="snow" )
        return options.get()


    def my_show_(*args): ##function called when options value changes
        str_out1.set(options1.get())
        return options1.get()


def my_show2(*args): ##function called when options value changes
    str_out2.set(options2.get())

    with open(yaml_file) as file:

        fruits_list = yaml.load(file, Loader=yaml.FullLoader)

    list2 = []
    for item, doc in fruits_list.items():
            if item == 'movements':
                for x,y in doc.items():
                  if x == options2.get():
                      if isinstance(y[0], list):
                          len_list = len(y[0])
                          for i in range(len_list):
                              list2.append(y[0][i][0])
                              list2.append(y[0][i][4])


    for i in range(1):
       for j in range(2, 3):
            window.columnconfigure(j, weight=1, minsize=100)
            window.rowconfigure(i, weight=1, minsize=100)
            frame_5 = tk.Frame(
                master=window,
                highlightbackground="black", highlightthickness=1
            )
            frame_5.grid(row=i, column=j, padx=5, pady=5)

            if i == 0 and j == 2:
                my_list_3 = list2
                options3 = tk.StringVar(frame_5)
                options3.set('Movement info') # default value
                om3 =tk.OptionMenu(frame_5, options3, *my_list_3)
                om3.grid()
                om3.config(bg="gray30", fg="snow", width = 10)
                str_out3=tk.StringVar(frame_5)
                str_out3.set("Output3")

    return options2.get()
```

Figure 15: Example of function called when drop down menu is pressed

Whenever the button 'Zur gespeicherten position' is selected on the saved positions for selected waypoint is called and passed through movej or movel command depending upon the selection made through GUI as shown in function go_to_saved_position() in figure 16. Similarly when the button 'Aktuelle Roboterposition speichern' the current position of the robot is replaced in place of already saved positions for the selected waypoint as shown in figure 16.

```
#Function to make robot move
def go_to_saved_position():
    point_name = my_show()
    _ , _,current_point_data = yaml_data()
    joint_pos = current_point_data[point_name][0]
    endeff_pos = current_point_data[point_name][1]

    #move robot to saved point
    if options1.get() == 'movej':
        rob.movej(joint_pos, a, v, True)
    else:
        rob.movel(endeff_pos, a, v, True)


#Function to save the current robot position
def replace_and_save_current_position():
    with open(yaml_file) as file:
        fruits_list = yaml.load(file, Loader=yaml.FullLoader)


    for item, doc in fruits_list.items():
        if item == 'positions':
            replace_and_save_current_position.temp = doc[my_show()]
            replace_and_save_current_position.temp_point = my_show()
            doc[my_show()] = [[round(val, 4) for val in rob.getj()], [round(val, 4) for val in rob.getl()]]


    with open(yaml_file, 'w') as file:
        yaml.dump(fruits_list, file,sort_keys=False)


    _, _, current_point_data = yaml_data()
    label_3_1_6.config(text = current_point_data[options.get()][0])
    label_3_1_3.config(text = current_point_data[options.get()][1])
    label_3_2_8.config(text = text_update_position,bg="gray30", fg="snow" )
    file.close()
```

Figure 16: Additional functions in the script

4. Task to program PLC:

During my Internship, I was tasked to program and develop logic on PLC (SICK PLC) for lighting Red, Blue, Green, and White LED lights in the given order with a time of 2 seconds for each LED. Below is the picture showing the logic created to achieve this goal.

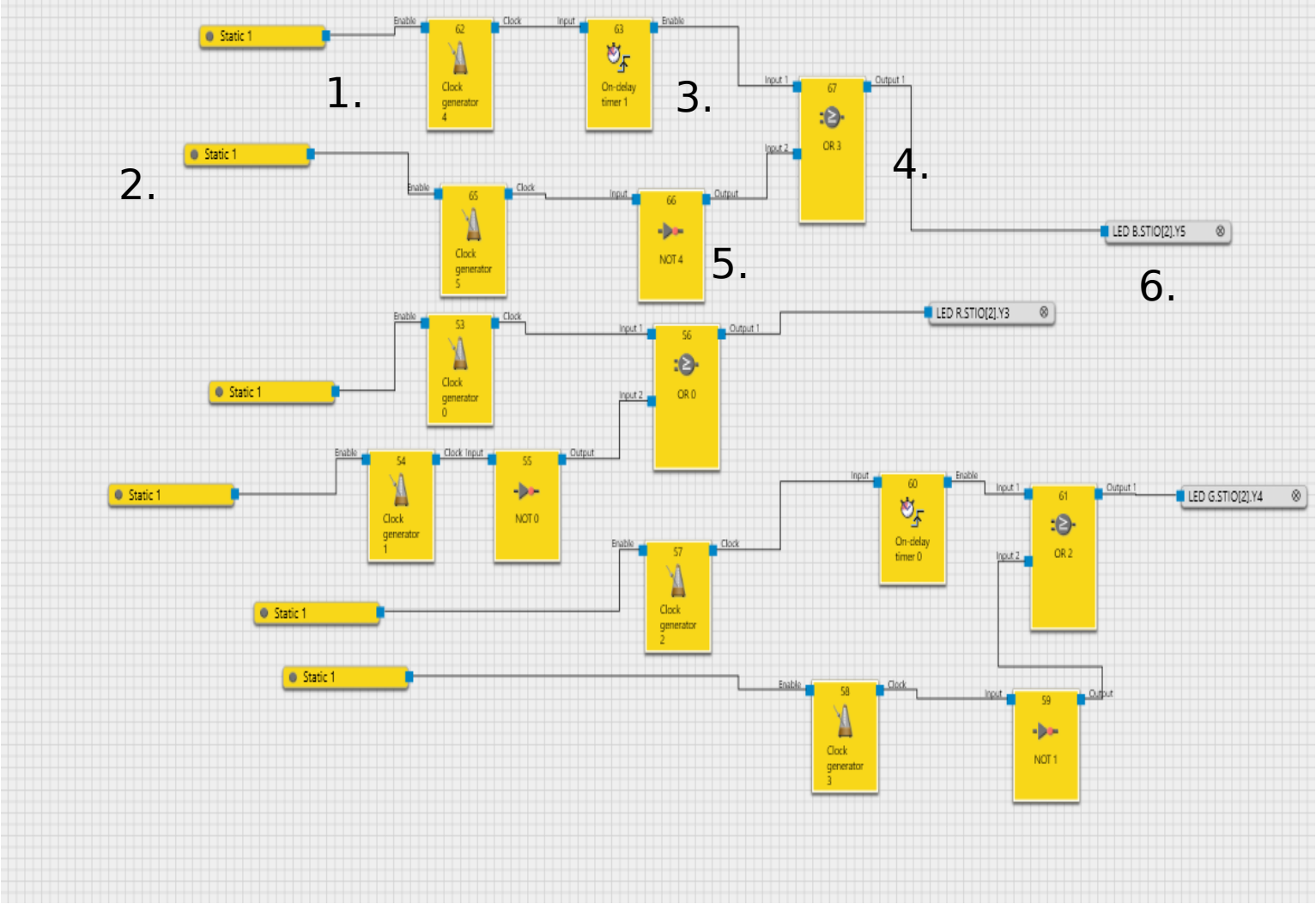


Figure 17: picture of Block programming in PLC to create the logic

1. The Clock generator function block allows to generate a pulsed signal. When the Enable input is set to High, the Clock output pulsates from Low to High and back to Low in accordance with the function block parameter settings. The Clock output switches to Low when the Enable input is set to Low. Figure 18 mentions the parametrs of clock generator.

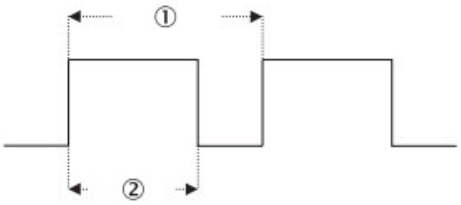


Figure 18: Clock generater paramters, (1)Elementary period, (2) Pulse time

2. This tab enables a static high signal.

3. The On-delay timer function block delays activation of the Enable output by a configurable period of time.

4. In OR block, the output is set to High when any of the evaluated inputs is High.

5. In NOT block, the value at the output is the inverted value of the input. If, for example, the input is set to High, the output is set to Low.

6. This tab represent differnt LED outputs where R is for Red, G is for Green , B is for Blue.

Note: For white light all of the 3 lights were on

5. Task to add Forward kinematic script to Robot:

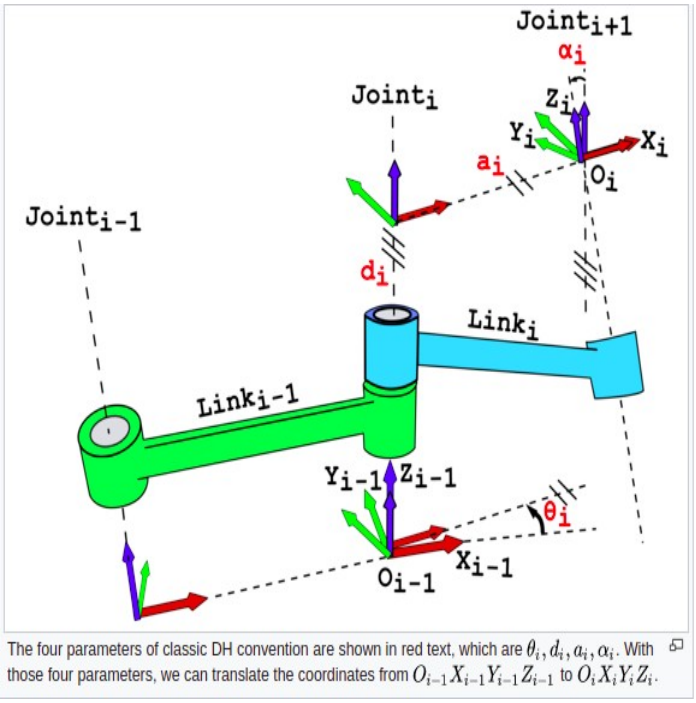
As discussed several waypoints in Sortbot, one parameter for these waypoints is blend radius. If the sum of the blend radius of two consecutive points exceeds the total distance between them then setting blending radius does not give the correct motion of the end effector.

To deal with this issue when we are given joint positions for two consecutive points we need to convert joint positions to end-effector positions and then calculate the absolute difference between these consecutive points. For this, I added a forward kinematic script to the UR python driver so that when we get the issue of blending radius higher than the distance between two points we get an error message.

DH Parameter for UR10:

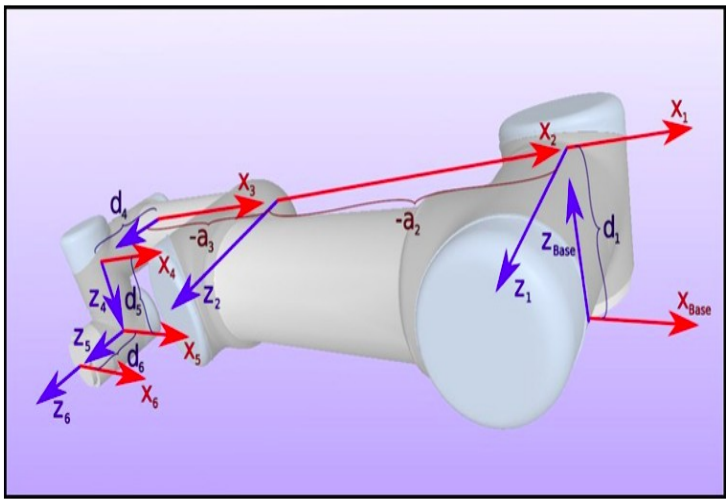
In mechanical engineering, the Denavit–Hartenberg parameters (also called DH parameters) are the four parameters associated with a particular convention for

attaching reference frames to the links of a spatial kinematic chain, or robot manipulator [SOURCE].



The four parameters of classic DH convention are shown in red text, which are $\theta_i, d_i, a_i, \alpha_i$. With those four parameters, we can translate the coordinates from $O_{i-1}X_{i-1}Y_{i-1}Z_{i-1}$ to $O_iX_iY_iZ_i$.

(a)



(b)

UR10							
Kinematics	theta [rad]	a [m]	d [m]	alpha [rad]	**Dynamics**	Mass [kg]	Center of Mass [m]
Joint 1	0	0	0.1273	π/2	Link 1	7.1	[0.021, 0.000, 0.027]
Joint 2	0	-0.612	0	0	Link 2	12.7	[0.38, 0.000, 0.158]
Joint 3	0	-0.5723	0	0	Link 3	4.27	[0.24, 0.000, 0.068]
Joint 4	0	0	0.163941	π/2	Link 4	2	[0.000, 0.007, 0.018]
Joint 5	0	0	0.1157	-π/2	Link 5	2	[0.000, 0.007, 0.018]
Joint 6	0	0	0.0922	0	Link 6	0.365	[0, 0, -0.026]

(c)

Figure 19: (a)[SOURCE] and (b) describe visually the DH parametrs, (c) lists the DH parameter for UR10 [SOURCE].

The script for Forward kinematics [SOURCE] is shown below in Figure 21.

```
d = mat([0.1273, 0.0, 0.0, 0.163941, 0.1157, 0.0922])#ur10 mm

a =mat([0.0, -0.612, -0.5723, 0.0, 0.0, 0.0])#ur10 mm

alph = mat([pi/2, 0, 0, pi/2, -pi/2, 0 ]) # ur10


# ********************************************** FORWARD KINEMATICS

def AH( n,th,c):

  T_a = mat(np.identity(4), copy=False)
  T_a[0,3] = a[0,n-1]
  T_d = mat(np.identity(4), copy=False)
  T_d[2,3] = d[0,n-1]

  Rzt = mat([[cos(th[n-1,c]), -sin(th[n-1,c]), 0 ,0],
                [sin(th[n-1,c]),  cos(th[n-1,c]), 0, 0],
                [0,               0,              1, 0],
                [0,               0,              0, 1]],copy=False)


  Rxa = mat([[1, 0,                 0,                  0],
                        [0, cos(alph[0,n-1]), -sin(alph[0,n-1]),   0],
                        [0, sin(alph[0,n-1]),  cos(alph[0,n-1]),   0],
                        [0, 0,                 0,                  1]],copy=False)

  A_i = T_d * Rzt * T_a * Rxa


  return A_i

def HTrans(th,c ):
  A_1=AH( 1,th,c  )
  A_2=AH( 2,th,c  )
  A_3=AH( 3,th,c  )
  A_4=AH( 4,th,c  )
  A_5=AH( 5,th,c  )
  A_6=AH( 6,th,c  )

  T_06=A_1*A_2*A_3*A_4*A_5*A_6

  return T_06
```

Figure 20: Python script for Forward kinematics

TASKS RELATED TO SPLITBOT

1. Developing scripts to monitor real time data of Splitbot:

I was tasked to create the scripts to monitor real-time data of Splitbot and calculate its efficiency, after the data were collected they needed to be transferred to BMW central server called IPST for monitoring. Similar real-time data transfer is also done in Sort bot therefore I was able to use the same script used in Sortbot and adjust them according to the needs of Splitbot.

The task was divided into two broad parts:

1. To get the real-time data and calculate efficiency (performance tracking script)

2. To transfer the data to BMW servers (IPST script).

Further explanation of both of these parts is done below.

1.1 To get the real time data and calculate efficiency:

A ROS node was created for this purpose which will be launched whenever Splitbot starts.

Input to the script was current Robot mode, information about KLT depalletized on basis of empty or full. All of them are ROS parameters running on the actual robot which are called in the script. Furthermore, the input of maintenance mode and system shutdown was also included. The code for this input is shown in figure 21. The input “KLT_COUNTER” go through the script and with the help of a function updates the values of KLT depalletized so far till the time of day.

```
PARAM_DATA = {'ROBOT_MODE':'/splitbot/robot_mode',
              'KLT_COUNTER_EMPTY':'/splitbot/state_machine/info/depalletized_number/empty_klt',
              'KLT_COUNTER_FILLED':'/splitbot/state_machine/info/depalletized_number/full_klt'}

MAINTENANCE_PARAM =  '/splitbot/maintenance_mode'

SYSTEM_SHUTDOWN =  '/splitbot/shutdown'
```

Figure 21: Inputs to performance tracking script

After every 15 seconds (time set by us) the YAML file containing values shown in figure 22 is updated through the command set_yaml_config() function shown in figure 23 in which the current parameter like date, OEE (overall equipment efficiency), run time so far, an error occurred so far, klt depalletized [empty or full] and maintenance time is set.

```yaml
config_date: 2021.08.25
config_error_counter: 0
config_klt_counter_empty: 0
config_klt_counter_filled: 0
config_maint_time: 0
config_oee_orga: 100.0
config_oee_tech: 100.0
config_orga_err_time: 0.0
config_runtime: 0.00
```

Figure 22: Yaml file for real time data

```python
def set_yaml_config(self):
    '''
    set data to the rosparam server and save them inside yaml file
    '''
    loginfo('--- saving configuration: ---\noee_tech: %s\noee_orga: %s\ndate: %s\nruntime: %s\nabs_err_counter: %s\nklt_empty: %s\nklt_filled: %s',self.oee_tech,
        str(self.oee_orga), str(self.date_today), str(self.runtime_today), str(self.abs_error_counter), str(self.klt_abs_empty), str(self.klt_abs_filled))
    set_param(YAML_CONFIG['config_date'], self.date_today)
    set_param(YAML_CONFIG['config_oee_tech'], self.oee_tech)
    set_param(YAML_CONFIG['config_oee_orga'], self.oee_orga)
    set_param(YAML_CONFIG['config_runtime'], round(self.runtime_today, 2))
    set_param(YAML_CONFIG['config_error_counter'], self.abs_error_counter)
    set_param(YAML_CONFIG['config_klt_counter_empty'], self.klt_abs_empty)
    set_param(YAML_CONFIG['config_klt_counter_filled'], self.klt_abs_filled)
    set_param(YAML_CONFIG['config_orga_err_time'], round(self.orga_error_time, 2))
    set_param(YAML_CONFIG['config_maint_time'], round(self.maintenance_time, 2))
    #saving rosparam ipst data to yaml file
    call(PARAM_SAVE_COMMAND)
    #saving config timestamp
    self.config_timestamp = time()
```

Figure 23: Function to update the YAML file show in figure 22

Other calculations like Maintenance time, error time are also done in the script. The maintenance time and error time are useful to calculate the OEE (figure 24).

Two types of OEE are calculated here:

1. OEE Organizational
2. OEE Technical

Two types of error time are defined:

1. Technical error time: error occurred during depalletizing
2. Organizational error: after defined error time interval the extra time is counted in Organizational error.

```
def get_oee_tech(self, runtime, maintenance_time, klt_stacked, errors):
    '''
    returns calculated oee tech
    '''
    #self.logging.debug('--- get_oee_tech with runtime: {}, maint_time:{}, klt: {}, errors: {}'.format(runtime, maintenance_time, klt_stacked, errors))
    try:
        if runtime > maintenance_time and klt_stacked > 0:
            oee_tech = round((1.0 - float(maintenance_time)/float(runtime))*(1.0-float(errors)/float(klt_stacked))*100.0, 2)
        elif runtime > maintenance_time:
            oee_tech = round(((float(runtime) - float(maintenance_time))/float(runtime))*100.0, 2)
        else:
            oee_tech = 0.0
    except ZeroDivisionError:
        oee_tech = 0.0
        logwarn('--- oee_tech: system running time is at zero ---')
    if oee_tech > 100.0 or oee_tech < 0.0:
        logwarn('--- oee_tech value is outside logical borders: %s ---',str(oee_tech))
        oee_tech = 0.0
    set_param(DATA_VALUE_DICT['oee_tech'], oee_tech)
    return oee_tech

def get_oee_orga(self, runtime, error_time):
    '''
    returns calculated oee_orga
    '''
    try:
        if runtime > error_time:
            oee_orga = round((1.0-float(error_time)/float(runtime))*100.0, 2)
        else:
            oee_orga = 0.0
    except ZeroDivisionError:
        oee_orga = 0.0
        logwarn('--- oee_orga: system running time is at zero ---')
    if oee_orga > 100.0 or oee_orga < 0.0:
        logwarn('--- oee_orga value is outside logical borders: %s ---',str(oee_orga))
        oee_orga = 0.0
    set_param(DATA_VALUE_DICT['oee_orga'], oee_orga)
    return oee_orga
```

Figure 24: Functions to calculate OEE technical and organizational

If the day is changed then the values of YAML file shown above in figure 22 are reset to default values otherwise the real-time data Is displayed.

The data displayed in BMW main server (IPST) are:

1. OEE_tech: OEE technical
2. OEE_orga: OEE oranisational
3. Error_counter: error occurred so far

4. klt_counter_empty: Empty KLT depalletized so far

5. klt_counter_full: Full KLT depalletized so far

6. robot_mode_id: the current robot mode, i.e running, error, etc.

```
DATA_VALUE_DICT = {
    'oee_tech':'/ipst/data/oee_tech',
    'oee_orga':'/ipst/data/oee_orga',
    'error_counter':'/ipst/data/error_counter',
    'klt_counter_empty':'/ipst/data/klt_counter_empty',
    'klt_counter_filled':'/ipst/data/klt_counter_filled',
    'robot_mode_id':'/ipst/data/robot_mode_id'
}
```

Figure 25: Displayed data on IPST

As described, the script is Ros node, the script is initialized by init_node command, and then an object of the class is created for starting the script as shown in figure 26.

```
if __name__ == '__main__':
    init_node('perf_track_node', anonymous=False, log_level=INFO)
    client = performance_tracking()
    try:
        while not is_shutdown():
            try:
                client.main()
            except (KeyError, socket.error) as error:
                print('--- main error: {} ---'.format(error))
                sleep(2.0)
    except KeyboardInterrupt:
        client.close()
        quit()
```

Figure 26: Code to initialize the ROS node

1.2. Data Transfer to BMW server (IPST):

This script is used to transfer the data calculated in the performance tracking script (figure 25) to the IPST server of BMW. The connection is client-server type. The connection is made by Socket programming.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server [SOURCE].

Initially, the details for the IPST server for a particular plant location is input (Figure 27). The details contain IP address and port number to be used for communication.

```
SETTINGS_FILE = path.dirname(path.realpath(__file__)[:-10]) + '/ipst_settings.yaml'
PARAM_LOAD_COMMAND = ["rosparam", "load", SETTINGS_FILE, "/splitbot/ipst/"]
call(PARAM_LOAD_COMMAND)

location = get_param('/splitbot/location','BER')
sys_name = get_param('/splitbot/sys_name','M0LR3G01')


AKZ_SOURCE= get_param('/splitbot/ipst/'+location+'/'+sys_name+'/AKZ_SOURCE')
IPST_DEST = get_param('/splitbot/ipst/'+location+'/'+sys_name+'/IPST_DEST')
HOST = get_param('/splitbot/ipst/'+location+'/'+sys_name+'/HOST')
PORT = get_param('/splitbot/ipst/'+location+'/'+sys_name+'/PORT')

SERVER_ADDRESS = (HOST, PORT)
```

Figure 27: Input to set up the connection

As shown in figure 28. The arguments passed to socket() specify the address family and socket type. AF_INET is the Internet address family for IPv4. SOCK_STREAM is the socket type for TCP, the protocol that will be used to transport our messages in the network.

```
class socket_client():
    def __init__(self):
        #--- setup socket ---
        #AF_INET besteht aus IPv4, port
        #SOCK_STREAM setzt das TCP Protokoll zugrunde
        self.ips_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.ips_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.ips_socket.settimeout(SOCKET_TIMEOUT)
        #--- variables ---
        #flag if connection persists
        self.client_connected = 0

        #ping
        self.ping_timestamp = None

        self.value_param_dict_save = {}
        self.message_param_dict_save = {}
        #fill data dict with initial values
        for data_name in DATA_VALUE_DICT:
            self.value_param_dict_save.update({data_name:None})
            logdebug('add data value dict: {}'.format(self.value_param_dict_save[data_name]))
        for data_name in DATA_MESSAGE_DICT:
                self.message_param_dict_save.update({data_name:False})
                logdebug('add data message dict: {}'.format(self.message_param_dict_save[data_name]))
                #---------starting connection----------
        self.start_client()
```

Figure 28: Initializing the script and socket object

The Start_client function shown below in figure 29, establishes a connection with the client. By using the object created by socket.socket() shown in figure 28.

```
def start_client(self):
    #print('Client-name: %s' % self.ips_socket.gethostname())

    self.client_connected = 0
    try:
        loginfo('--- Start connection to server {} on port {} ---'.format(*SERVER_ADDRESS))
        self.ips_socket.connect(SERVER_ADDRESS)
        self.client_connected = 1
        loginfo('--- Connected successfully ---')
        sleep(2)
        #reset all messages initially
        for message in DATA_MESSAGE_DICT:
            self.send_message(';G' + message)
        self.send_message(';G' + ROBOT_OFF)
        self.send_message(';K' + ROBOT_ON)


    except socket.error:
        logwarn('--- Server not available ---')
        sleep(TAKTZEIT)


    except KeyboardInterrupt:
        sleep(TAKTZEIT)
        exit()
```

Figure 29: start_client function

For sending the message, the message is prepared as per the requirement by function send_message (figure 30) and sent through the _send_data function (figure 31) which uses sendall function to send the data.

```
def send_message(self, data, cycletime=TAKTZEIT):
    '''
    sending prepared messages as ipst 'datagram'
    '''
    ipst_message_header = AKZ_SOURCE+IPST_DEST+'MESSAGE_KEIN' + self._get_ipst_timestamp()
    ipst_message_footer = '_PLC'
    ipst_message_footer += data
    data_out = ipst_message_header + self._get_ipst_length(ipst_message_header + ipst_message_footer) + ipst_message_footer
    self._send_data(data_out)
    sleep(cycletime)
```

Figure 30: Adding data to message in required format

```
def _send_data(self, data_string):
    '''
    sending data over tcp/ipv4
    '''
    #raise error if length of String exceeded DATA_LENGTH
    if len(data_string) > DATA_LENGTH:
        logerr('--- output data:{}\n-string is exceeding maximum length: {} with max {}'.format(data_string, len(data_string), DATA_LENGTH))
    #add spaces to reach the desired length
    data_string = data_string.rjust(DATA_LENGTH, ' ')
    #logging.info('--- sending data: {} with length: {}'.format(data_string, len(data_string)))
    try:
        #sending data
        answer = self.ips_socket.sendall(data_string.encode('utf-8'))
        if answer is None:
            pass
            #logging.info(str(datetime.now()) + ' sending data successfully:\n{}'.format(self.data_out.encode('utf-8')))
        else:
            logwarn('--- sending data finished with errors: {} ---'.format(answer))
    except socket.error as error:
        logerr(error)
        self.client_connected = 0
        sleep(TAKTZEIT)
    except socket.timeout:
        logerr('--- socket timeout ---')
        self.client_connected = 0
        sleep(TAKTZEIT)
    except UnicodeDecodeError:
        logerr('--- sending message failed due to usage of special characters (ae, ue, oe, ss, ...), please remove them from the Text! ---')
        sleep(TAKTZEIT)
```

Figure 31: send_data function

This script is also a ROS node initiated when the Robot starts. Figure 31 shown below initializes the node and starts the client.

```
if __name__ == '__main__':
    rospy.init_node('ipst_node', anonymous=False, log_level=INFO)
    client = socket_client()
    try:
        while (not is_shutdown()):
            #Datenaustausch und Verarbeitung
            if client.client_connected:
                client.data_handling()
            else:
                client.start_client()
    except KeyboardInterrupt:
        client.close()
        quit()
```

Figure 32: ROS Node initialization

2. Automated testing of IPST Script:

Before testing the Performace tracking script and data transfer script in the real production environment, I tested the script through automated bash scripting by simulating robot conditions i.e creating errors, depalletizing scenarios for both empty and full KLTs. Figure 33 shows the bash script. test_4.py (figure 34) is used to simulate the Robot in different situations.

```
#!/bin/bash


echo First Group:
source /opt/ros/noetic/setup.bash
wait


echo second Group:
roscore &
(sleep 10 ; gnome-terminal -e 'sh -c "python3  perf_track_Splitbot.py; exec bash"') &
(sleep 9 ; rosparam set /splitbot/robot_mode 'RUNNING') &
(sleep 20 ; gnome-terminal -e 'sh -c "python3  test_4.py; exec bash"')
```

Figure 33: Automated Bash script to test the scripts

```
x = 0
y = 0
for i in range(17):
  if i % 5 == 0 and i != 0:
    if i == 5:
        set_param('/splitbot/robot_mode', 'PROBLEM')
        set_param('/splitbot/IH_Ruf', False)
        set_param('/splitbot/On_IPC_UR' , True)
        sleep(30)

    if i == 10:
        set_param('/splitbot/robot_mode', 'PROTECTIVE_STOP')
        set_param('/splitbot/IH_Ruf', False)
        set_param('/splitbot/On_IPC_UR' , True)
        sleep(30)

    if i == 15:
        set_param('/splitbot/robot_mode', 'EMERGENCY_STOP')
        set_param('/splitbot/IH_Ruf', False)
        set_param('/splitbot/On_IPC_UR' , True)
        sleep(10)

  elif(i % 2 == 0):
    if i == 6:
        set_param('/splitbot/robot_mode', 'PROBLEM')
        set_param('/splitbot/IH_Ruf', False)
        set_param('/splitbot/On_IPC_UR' , True)
        sleep(10)

    elif i == 8:
        y += 1
        set_param('/splitbot/robot_mode', 'RUNNING')
        set_param('/splitbot/state_machine/info/depalletized_number/full_klt', y)
        set_param('/splitbot/IH_Ruf', False)
        set_param('/splitbot/On_IPC_UR' , True)
        sleep(6)

    elif i == 16:
        set_param('/splitbot/On_IPC_UR' , False)

    else:
        x += 1
        set_param('/splitbot/robot_mode', 'RUNNING')
        set_param('/splitbot/state_machine/info/depalletized_number/empty_klt', x)
        if i == 4:
          set_param('/splitbot/IH_Ruf', True)
        else:
          set_param('/splitbot/IH_Ruf', False)

        set_param('/splitbot/On_IPC_UR' , True)
        sleep(6)
```

Figure 34: Script to create the Robot modes and situations

TRAINING DURING INTERNSHIP

1. Programming robot for Pick and Place application:

I learned how to program UR10 robot. I worked on application of pick and place using a Vaccum gripper. Below I have explained the command used.

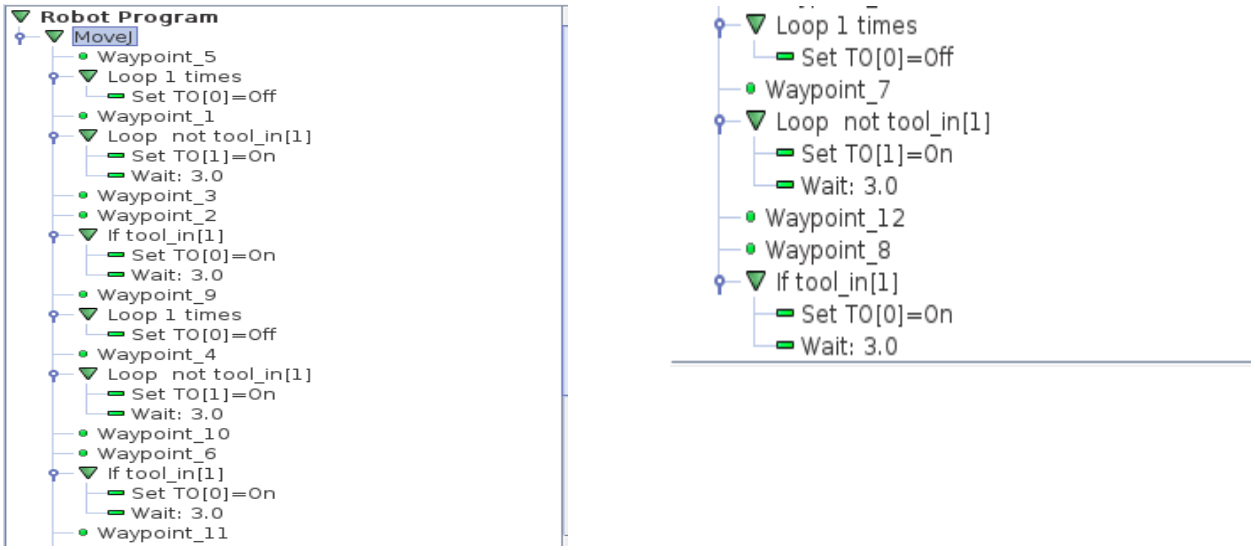


Figure 35: Pick and place programe

1. MoveJ command is used to move to requested joint positions. MoveL which is not used in this program is used to move to requested end-effector position and orientation.

2. Waypoint_X is used to set waypoints for pick and place application.

3. Loop X times is used to loop the command for X number of time.

4. Set is used here to On or Off the vaccum of gripper.

5. Wait is used to set a delay or wait time during program.