MTRN3060 Practical Logbook

Week 1:

TASK 2

2.2. The jog joint function rotates the corresponding joint about its axis. Jogging joint 0 rotates the base, jogging joint 6 rotates the robot wrist etc.

2.4. The jog linear function moves the coordinate of the end effector linearly to a given point. This adjusts the joints automatically to reach the given point, similar to how it would reach a point defined in a workobject.

2.6. Jog reorient is rotates the end effector coordinates. This alters the joint position to achieve the desired position.

TASK 3

Motion Mode	Joystick	Joint
Axis 1-3	X	Joint 1 jog
	Y	Joint 2 jog
	Z	Joint 3 jog
Axis 4-6	X	Joint 4 jog
	Y	Joint 5 jog
	Z	Joint 6 jog
Linear	X	Linearly moves end effector along X axis
	Y	Linearly moves end effector along Y axis
	Z	Linearly moves end effector along Z axis
Reorient	X	Rotates end effector coordinates about X axis
	Y	Rotates end effector coordinates about Y axis
	Z	Rotates end effector coordinates about Z axis

TASK 4

Type	DOF	Speed	Accuracy	Flexibility	Cost (USD)
Articulated	4 or more	Approx. 450deg/s max	+- 1mm	High, depending on DOF	Wide range (10k – 150k)
SCARA	4	Very fast (up to 7m/s)	+-0.01mm to <10microns	Limited payload mainly pick and place	Varies with size 25k – 400k
Delta	4 (X, Y, Z and end effector)	Fast due to low weight	+- 0.1mm	Limited payload limited motion	As low as 5k
Cartesian	3 DOF	5m/s or more	Very accurate	Cannot reach around obstruction	As low as 1k
Cylindrical	3 DOF	1000m/s to 10,000m/s	Average. Depends on hardware	Many applications but low range of movement	5k to 15k
Polar	3 DOF depend on	Relatively slow	Relatively inaccurate	Large footprint large payload	Expensive given size

Aspect	**Industry Robot (IRB)**	**Collaborative Robot (CRB**
Safety	Requires training, risk assessment, lockout procedures	Built in safety with sensors and cameras. Can adjust speed and position based on sensor input.
Speed	Max under 1m/s	Fully collaborative robots move about 250mm/s
Task	Whatever the robot is designed for	Whatever the robot is designed for
Programming	Coding (C++ RAPID Java Python)	Learn by observation of humans completing tasks
Flexibility	Low, requires reprogramming	High, can be taught new tasks easily
Accessibility	Low, requires training and expertise	High, can be used safely by anyone with minimal training.
Performance	High performance	Low performance for safety

Week 2:

Feature	**Revolute**	**Linear**	**Twisting**	**Orthogonal**	**Revolving**
Structure	Pin or knuckle joint through a rotary bearing	Rail or telescoping mechanism and piston	Two shafts connected with a bearing parallel	Rail with a output link running perpendicular to it	Two shafts connected with a bearing perpendicular
Mobility	One degree of freedom (rotation)	One degree of freedom (linear)	One degree of freedom (rotation)	One degree of freedom (linear)	One degree of freedom (rotation)
Example	Elbow, knee, interphalangeal joints	Gas lift in a chair	Wheel on a skateboard	Overhead power lines for trains	Propeller and hub

No	**Type**	**Mobility**
0	Twisting	1 Degree of freedom
1	Revolute	1 Degree of freedom
2	Revolute	1 Degree of freedom
3	Twisting	1 Degree of freedom
4	Revolute	1 Degree of freedom
5	Twisting	1 Degree of freedom

2.1. The IRB120 has a roughly spherical work envelope with a gap at the back where it would interfere with its own connections. It also has a gap near the base of the robot where it would collide with itself.

2.2 The IRB120 uses polar configuration, as it has a spherical work envelope based around a fixed base point.

2.3 The end effector coordinate is offset perpendicular to the flange by 20cm.

Week 3:

Here are highlights of this week's practical:

- Introduced to RAPID programming and became familiar with paths and programs in the RobotStudio environment.
- Learned to mount tools to the end of the IRB120
- Learned how to use autopath to generate paths around geometry perimeters.
- Learned how to transfer RAPID code to the real IRB120
- Experimented with creating a new pen holder in CAD

Week 4:

Using MHS3-20D

1.4.1. What is the Bore dimension? What does the Bore dimension mean?

20mm, the "bore dimension" refers to the diameter of the central hole or opening within the gripper's body.

1.4.2. What is the maximum Pressure and Internal Force of 20mm fingers?

0.6 MPa pressure and approximately 36N

1.4.3. Why are there two locations for mounting the Auto switch?

Provides flexibility of mounting options for sensor placement. Different applications will require different sensing capabilities.

1.4.4. How to mount the adapter to the robot flange?

Given "The adaptor and fingers are customized parts for each Robot and tool" the mounting procedure differs for each custom adapter. Although typically it involves passing fasteners through the adapter into the appropriate mounting holes in the adapter, and into the mounting holes on the robot.

1.4.5. How to mount the MHS3-20D gripper to the adapter?

This process varies from adapter to adapter.

Week 5:

EEPROM

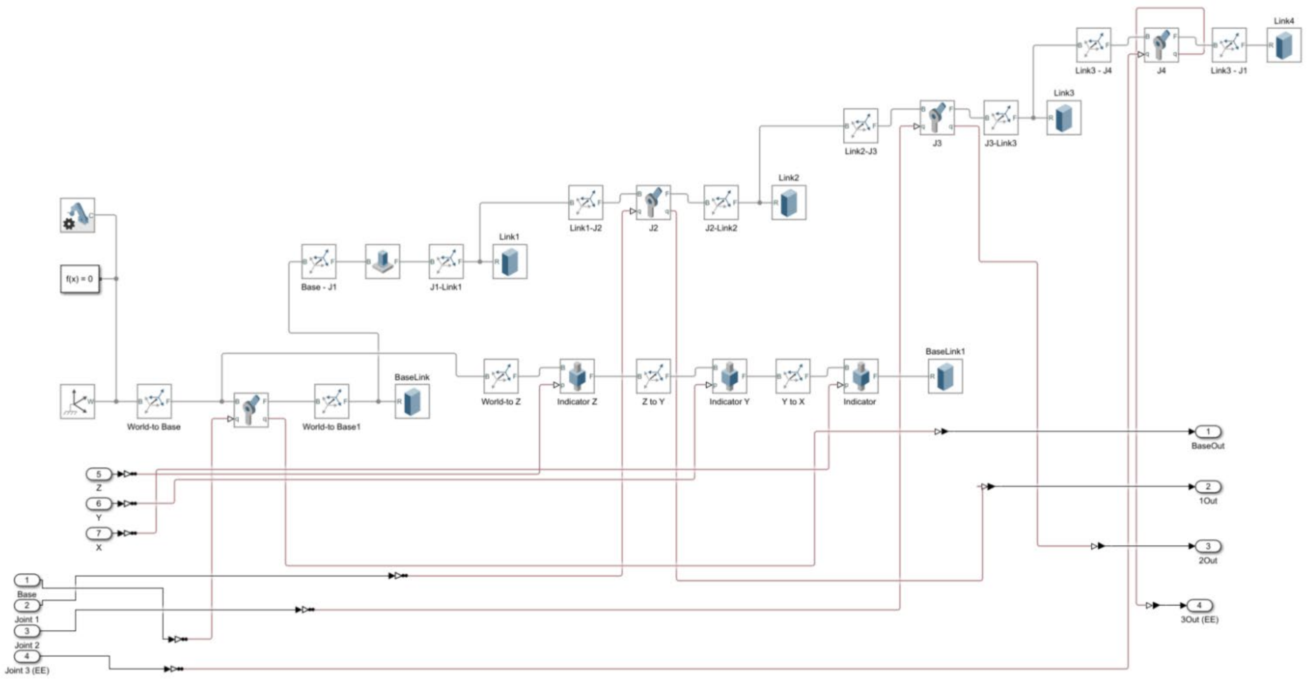
Address	Data/Function Name	Usage	Access	Initial Value
0	Model Number (L)	Lowest byte of model number	R	12
1	Model Number (H)	Highest byte of model number	R	0
2	Version of Firmware	Info on version of firmware	R	-
3	ID	ID of dynamixel	RW	1
4	Baud Rate	Baud rate of dynamixel	RW	1
5	Return Delay Time	Return delay time	RW	250
6	CW Angle Limit(L)	Lowest byte of clockwise angle limit	RW	0
7	CW Angle Limit(H)	Highest byte of clockwise angle limit	RW	0
8	CCW Angle Limit(L)	Lowest byte of CC angle limit	RW	255
9	CCW Angle Limit(H)	Highest byte of CC angle limit	RW	3
11	Highest Limit Temp	Internal limit temp	RW	70
12	Lowest Limit Voltage	Lowest limit voltage	RW	60
13	Highest Limit Voltage	Highest limit voltage	RW	140
14	Max Torque (L)	Lowest byte of max torque	RW	255
15	Max Torque(H)	Highest byte of max torque	RW	3
16	Status return level	Status return level	RW	2
17	Alarm LED	LED for alarm	RW	36
18	Alarm Shutdown	Shutdown for Alarm	RW	36

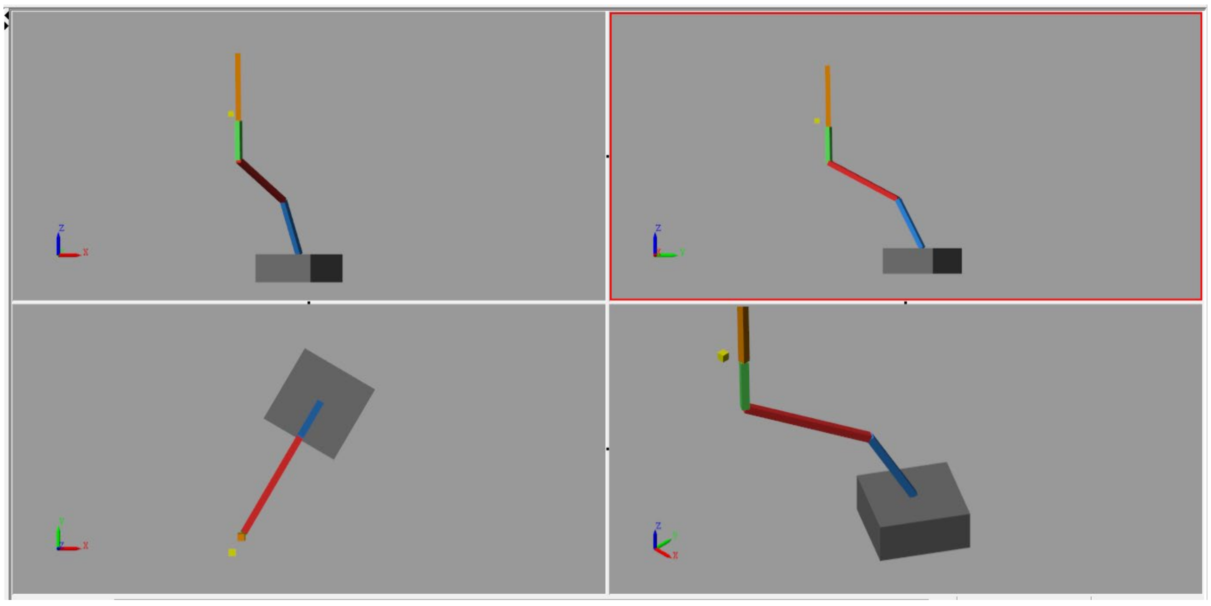
RAM

Address	Data/Function Name	Usage	Access	Initial Value
24	Torque Enable	Torque on/off	RW	0
25	LED	LED on/off	RW	0
26	CW Compliance Margin	CW Compliance Margin	RW	1
27	CCW Compliance Margin	CCW Compliance Margin	RW	1
28	CW Compliance Slope	CW Compliance Slope	RW	32
29	CCW Compliance	CCW Compliance Slope	RW	32
30	Goal Position L	Lowest byte of goal position	RW	
32	Goal Position H	Highest byte of goal position	RW	
34	Torque limit L	Lowest byte of torque limit	RW	ADD14
35	Torque limit H	Highest byte of torque limit	RW	ADD15
36	Present position L	Lowest byte of current position	R	
37	Present position H	Highest byte of current position	R	
38	Present speed L	Lowest byte of current speed	R	
39	Present speed H	Highest byte of current speed	R	
40	Present load L	Lowest byte of current load	R	
41	Present load H	Highest byte of current load	R	
42	Present voltage	Current voltage	R	
43	Present temp	Current temp	R	
44	Registered	Means if instruction is registered	R	0
46	Moving	Means if there is any movement	R	0
47	Lock	Locking EEPROM	RW	0
48	Punch L	Lowest byte of punch	RW	32
49	Punch H	Highest byte of punch	RW	0

Week 6:

A model is required to properly simulate the 4 DoF (Degrees of Freedom) robotic arm for the major project. This requires a structure, and a kinematic model to determine the transformation matrices and position of each joint. The model demonstrated in the practical document is unsuitable as it lacks the degrees of freedom needed. The structure assembled for the major work is shown below:

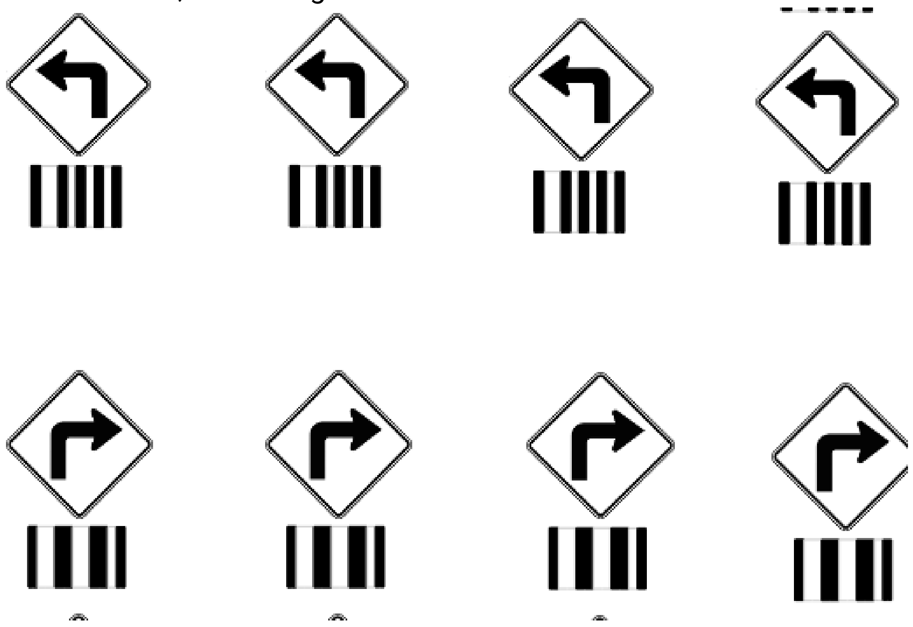




The blue member of the arm is fixed at an angle, as the first axis of rotation is offset from the base at an angle. The base is capable of rotating about the Z axis of the world frame, and each joint articulates similarly. In this example, an orange block is used to denote the end effector, this was modelled this way for simplicity.

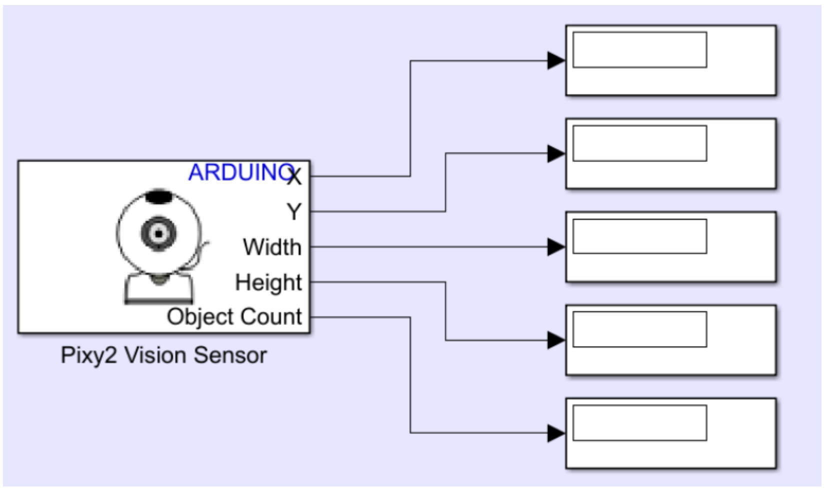
Week 7:

This practical had us use a Pixycam to identify different coloured blocks. Using it in conjunction with an Arduino to perform simple shape and colour recognition. The pixy cam was able to be trained to recognize these barcodes. A team for the major project was formed, consisting of Mackenzie Holden and Rahul Das.



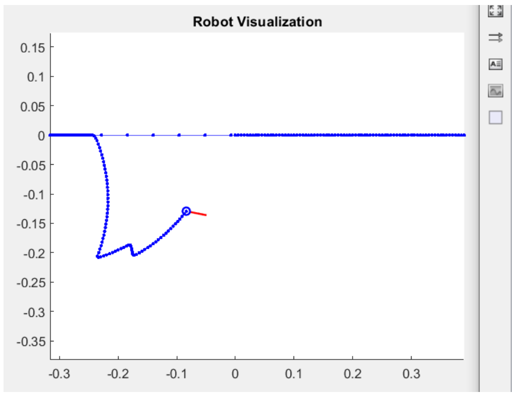
An issue arose with the number of matches being detected by the camera. The camera would only pick up 3 different matches at a time, struggling to pick up the fourth. This was resolved by tweaking exposure, white balance, and image recognition parameters to accurately pick up each unique object.

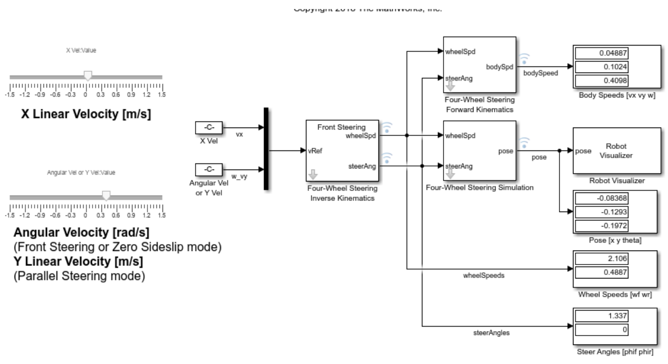
I attempted to implement this into Simulink in order to use it with the robotic arm. By itself the camera worked well, however combined with controlling the arm and performing kinematics calculations Matlab became too unstable and crashed frequently. Lab computers are more powerful but have outdated and incompatible versions of Matlab installed. The processing power required to do this with one system and the delay in communications is too great. More work is needed to make the system more stable.



Week 8:

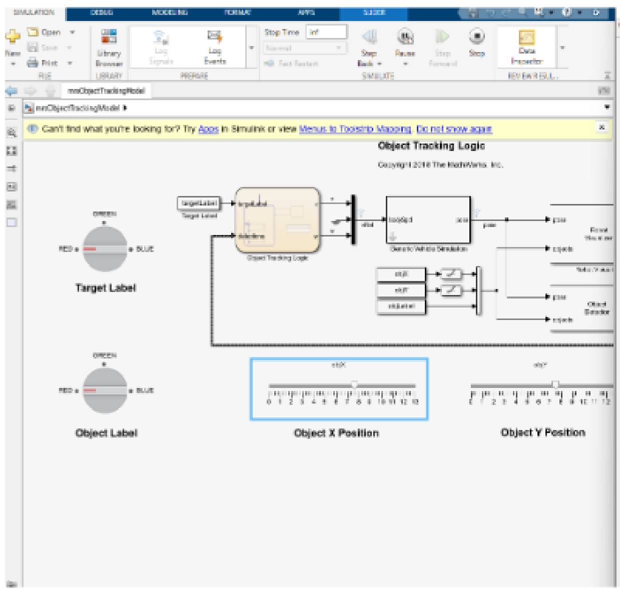
This practical involved mobile robot kinematics, and different types of four-wheel control. The model this robot was designed for is front-wheel drive, however the turning circle was too large and as a result is difficult to control. Using differential drive allowed the robot to turn while remaining relatively stationary in the X Y direction. The practical recommends four-wheel steering with only 2 wheels being moved as per the construction of the robot, more work will be needed to determine the best implementation.

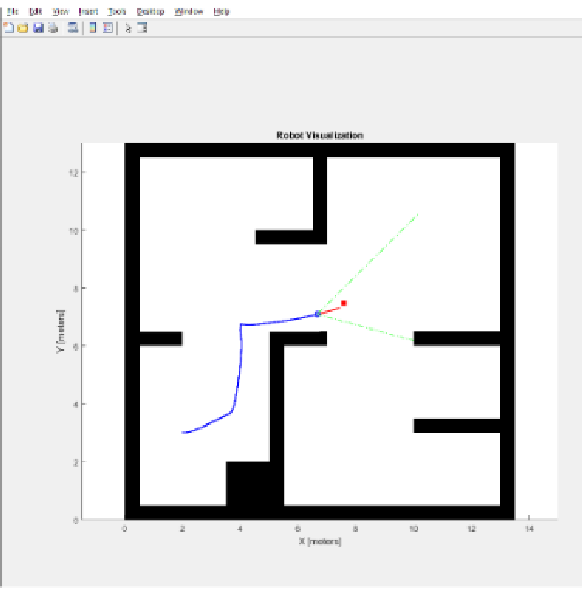




This is an example of 4 target following for a simulated model of 4 wheel driven robot with front steering. The top slider controls the speed of the robot, and the bottom slider controls the angle of the front of the robot.

We also simulated target following robots, the simulation and construction is shown below.





Week 9 and Week 10:

Week 9 was spent working on the belt with the arm equipped, while week 10 was spend working with the conveyor belt and IR sensor combination.

The belt arm was controlled with a simple Arduino sketch, the function used to move the belt is shown below:

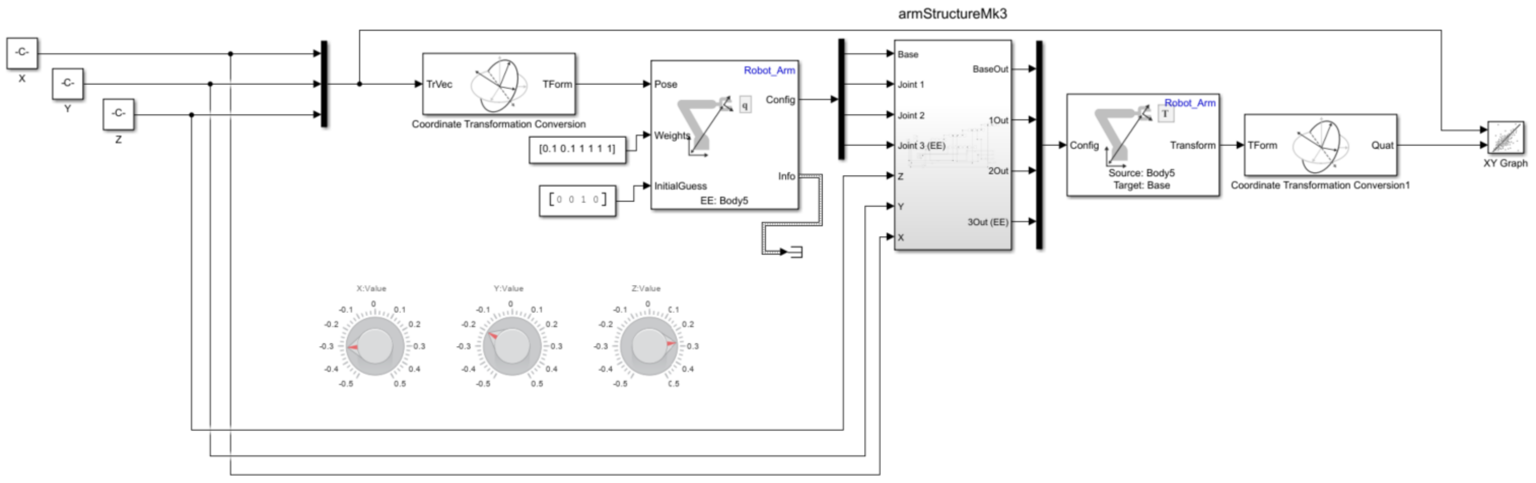
```
void moveSteps(signed long steps)
{
 if (steps > 0)
 {
   armPos = armPos + steps;
     for (int i=0; i<steps; i++)    //Forward steps
       {
       digitalWrite(DIR,LOW);
       digitalWrite(ENA,HIGH);
       if (switchState == HIGH){break;}
       digitalWrite(PUL,HIGH);
       delayMicroseconds(Del);
       digitalWrite(PUL,LOW);
       delayMicroseconds(Del);
       }
 }
 /////////////////////////////////////
 if (steps < 0)
 {
   steps = abs(steps);
   armPos = armPos - steps;
     for (int i=0; i<steps; i++)   //Backward steps
     {
       digitalWrite(DIR,HIGH);
       digitalWrite(ENA,HIGH);
       if (switchState == HIGH){break;}
       digitalWrite(PUL,HIGH);
       delayMicroseconds(Del);
       digitalWrite(PUL,LOW);
       delayMicroseconds(Del);
     }
 }
}
```

This code was combined with a limit switch to determine the zero position. Using this accurate position control was possible by determining the relationship between steps and millimetres moved. The belt position was zeroed every cycle to retain accuracy.

The belt with the IR sensor was very similar, with the main difference being the code to detect objects in front of the sensor. When the sensor was triggered, in this case by a cube, the belt would stop. This was important for the major project, as we determined this would be one of the simplest ways to implement as a solution.

Week 11:

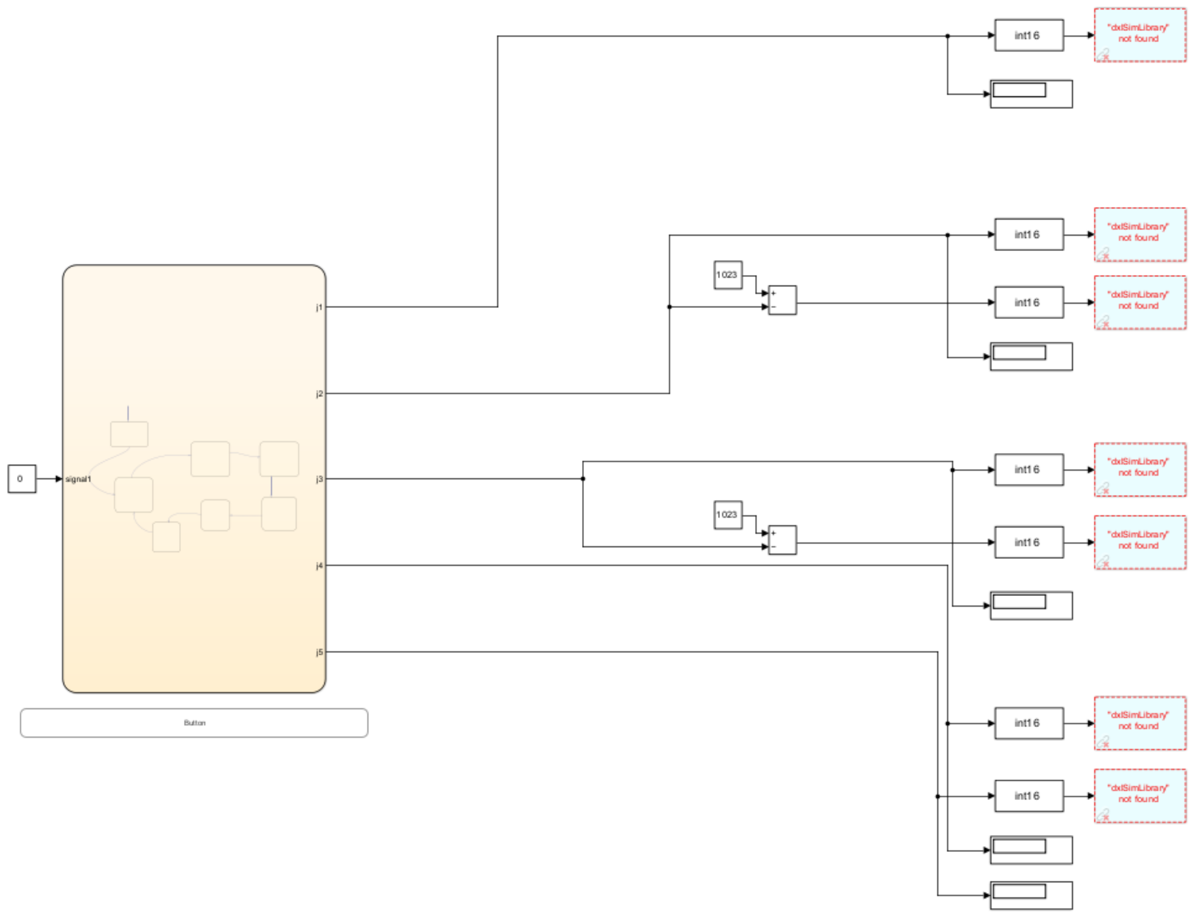
Within our team of three I was working on the arm model and simulation, the other members worked on the car simulation and IR sensor belt combo. During this time, we determined that the belt and IR sensor combined with an arm on one side would be suitable for the major project. More work was completed on the simulation with inverse kinematics. I was able to accurately implement inverse kinematics in simulation but struggled with interfacing the simulation to the physical arm. This was made more difficult with the minimal access to the hardware, however this is a known constraint. Ideally the inverse kinematics model would be used with a Stateflow diagram to determine the X, Y, and Z coordinates of the target and move the joints to match it. In the model below dials are used to manage the XYZ coordinate of the target, represented by a yellow cube.



While the simulation by itself worked, implementing it in practice was a challenge, this will need more development before it is implemented into the final project.

Week 12

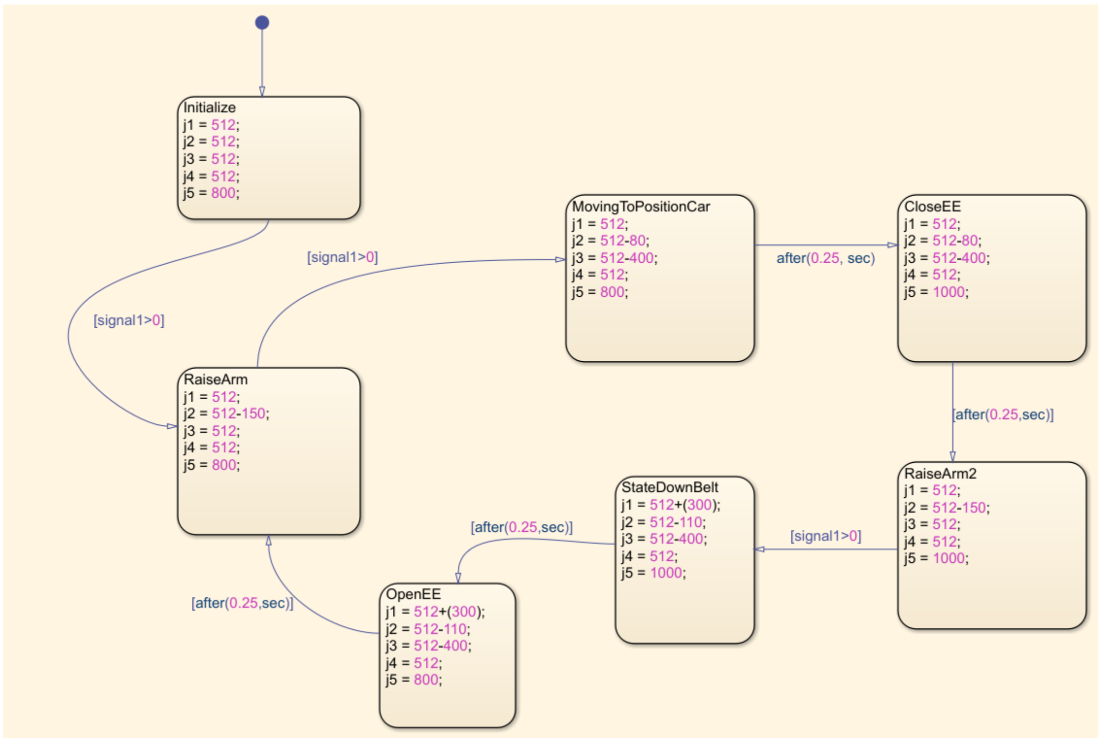
As time constraints became apparent, it was clear the inverse kinematics simulation would not be ready in time to be implemented into the final project. In order to produce a functional system, a stateflow diagram would be used to directly control the angle of each joint. The structure of this control system is shown below:



The joint outputs are to be determined by the angle required. This will need to be calibrated in person, in conjunction with the datasheet for the AX-12A motors being used for this application. The stateflow diagram is driven using a push button to progress through each state of the diagram. Ideally, this would be replaced with the input from a sensor or from the mobile robot model when the robot is in position. Integrating these systems together is time and resource intensive computation wise. With a more powerful computer it may be more feasible, this currently needs development.

Week 13:

The majority of this week's practical was spent tuning the stateflow diagram to achieve the required results. The final stateflow diagram for the submission is shown below:



In this example signal1 indicates the input from the button, when the button is pressed, we move states, some states are controlled by delays. These should be delays of approximately 2 seconds in real time, however Matlab runs much slower than real time often taking multiple seconds for one second in simulation time. This was another symptom of having an underpowered computer for the system, as the pacing should be one to one.

First the arm initialized in a state where every member is extended fully, and the end effector is open. This moves into a Raised arm position which will be the default state for the system when not moving. Pressing the button sets our state to MovingToPositionCar, where the arm positions itself to pick up the block. After a delay the end effector closes then moves to RaiseArm2, which holds the block in the raised position. Pressing the button again will move to StateDownBelt which positions the arm over the belt, then to OpenEE which opens the end effector releasing the block. This then moves back to the RaiseArm position, ready for the next button press to pick up the block.

This worked in practice, however many improvements can be made. The main challenge with the project was integration. Making multiple systems communicate quickly, accurately and autonomously is difficult with limited computing power. Matlab would regularly crash even running this simple motor control state diagram, and would crash almost every time an implementation of inverse kinematics was used. With more knowledge of the way Matlab performs calculations this could potentially be avoided, time constraints made this difficult. Using a PixyCam with colour recognition would be able to move the arm to pick up the block regardless of position on the mobile robot, proximity sensors could mounted to the arm or the mobile robot to determine when the car arrives. These could be implemented, however would slow the system without a powerful enough processor.