LONG TERM GOAL

1. Create ICub robot inside webots+ Environment. Press the button -movable

2. Create a PMP class/library that offers, a PMP for any bodychain, and with multiple parallel goals [**Inverse kinematics**]
3. Allows control for desired **pose** x, **velocity** x_dot, **configuration** q, joint velocities, q_dot, **force**, joint **torques**,
4. Allows this for any robot given **forward Kinematics** (fK), **Jacobian** J, Stiffness K, Admittance A)
5. Ikpy library python or rtb
6. https://github.com/BhatAjaz/morphoGen/tree/master/src/control/PMP/src

```
class PMP:
    x, x_dot, q, q_dot, F, tau = []
    FKin, Jac, K_stiff, A_admittance = []
    TBG = []
    def stateUpdate(self):
        #update the above variables from robot
        pass
    def setGoal(self):
        # set any of the variables (say pose) as a goal for the system
        pass
    def setRobotBody(self):
        #specify robot specific details of FKin, Job, K_stiff, A_admittance
        pass
    def step(self):
        #does 1step of PMP simulation on the body chain specified
        Pass
```

```
class PMP:
    x, x_dot, q, q_dot, F, tau = []
    FKin, Jac, K_stiff, A_admittance = []
    TBG = []

    def stateUpdate(self):
        # update the above variables from robot
        pass

    def setGoal(self):
        # set any of the variables (say pose) as a goal for the system
        pass

    def setRobotBody(self):
        # specify robot specific details of FKin, Job, K_stiff, A_admittance
        pass

    def step(self):
        # does 1 step of PMP simulation on the body chain specified based on
concepts in this passage
```

```
        # Step 1: Generate a target-centered, virtual force field in the extrinsic space
        F = K_ext * (x_t - x)

        # Step 2: Map the force field from the extrinsic space into virtual torque field in the intrinsic space
        T = Jac_T * F

        # Step 3: Relax the arm configuration to the applied field
        q_dot = A_int_T * T

        # Step 4: Map the arm movement into the extrinsic workspace
        x_dot = Jac * q_dot

        # Step 5: Integrate over time until equilibrium
        x = x + (x_dot * dt)

        # update state variables
        self.x = x
        self.x_dot = x_dot
        self.q = q
        self.q_dot = q_dot
        self.F = F
        self.tau = T
        self.TBG = TBG
```

https://github.com/JunaidFarooqZargar/robotics1
https://arxiv.org/pdf/1805.07263.pdf

Date/Week: Week 27 (7/11/2023)

Objective 1: Finish Report Draft
Objective 2: Prepare for Presentation

Discussion /Lessons Learnt:

7/11/2023 Meeting

Discussed Presentation Material

Discussed and reviewed what to include for the report, told to focus on:
Highlight the library development and how this will not work for 3D robot and make it understandable

Overall Progress:

Presentation Given and Draft Submitted

Getting ready for Final Code Review

Next Week Objectives:

Date/Week: Week 26 (31/10/2023)

Objective 1: Finish Report Draft
Objective 2: Refractor Code
Objective 3: Prepare for Presentation

Discussion /Lessons Learnt:

31/10/2023 Meeting

Discussed wrapping up project, told to document project and present it in way that a new student could easily take over. Highlighting the class and the purpose, along any difficulties faced.

Told to modify existing planar and create variations of them for the presentation.

Regarding changes to Code:

Modified calculate_error_vector function to work out numpy library ,Now the function is manually iterating through the elements of the setpoint_X and current_X lists and performing the subtraction for each element

Modified Modified calculate_error_vector function to work out numpy, function now takes the A_int matrix and Torque vector as inputs, checks for compatibility, and then manually computes the dot product using nested loops. The result, q_dot, is a list of values.

This modified function takes the K_ext matrix and x_error vector as inputs, checks for compatibility, and then manually computes the dot product using nested loops. The result, force, is a list of values. This code replicates the functionality of NumPy's np.dot function without relying on the library.

Modified variable x_target in function pmp_kinematics to now be list containing the elements from target_position instead of using numpy array and can be used in most cases where you'd use a NumPy array for simple tasks.

Created a new function calculate_Rmse manually calculate the sum of squared errors, the mean of squared errors, and then take the square root to get the RMSE. Instead of using Numpy

This function is called in run_step function inside the while loop and replaces the line rmse = np.sqrt(np.mean(np.square(x_error))

Modified Modified calculate_Torque function to work out numpy function now can calculate the torque without using NumPy by manually implementing the matrix transpose and matrix-vector multiplication to obtain the torque. This approach replicates the functionality of the NumPy code you provided without using the library.

Overall Progress:

Test Cases are working for current version of the class
Removed all usage of numpy library in Pmp Class

Next Week Objectives:

Date/Week: **Week 25 (24/10/2023)**

Objective 1: Resolve the issue of why the joints of the robots in the four worlds are not working

Discussion /Lessons Learnt:

24/10/2023 Meeting

Robot was still unable to move

Purposed solutions to resolve:

- Update webots
- Remove all code and slowly import code

Solution

During attempt to create position sensors to check current positions of robot joints, added robot.step. This line resolves the issue of joints not moving and being set to the positions calculated by the class.

Worked on the Four Robots

- 2joint planar is working Completely
- 3joint planar is working Completely
- URE3 is partially working (no error returned by incorrect positions)
- Managed to configure code for the 6 joints of the URE3
- 3joint 3dof (same issue with URE3)
- Testing with Imu robot (no error returned by also incorrect position) to replace 3joint_3dof robot

Link to IMU Source

https://github.com/cyberbotics/webots/blob/released/projects/samples/devices/worlds/imu.wbt

Overall Progress:

Joint not moving Issue is Resolved

Next Week Objectives:

Date/Week: **Week 24 (17/10/2023)**

Objective 1: Resolve the issue of why the joints of the robots in the four worlds are not working

Discussion /Lessons Learnt:

17/10/2023 Meeting

The robots were unable to move during the meeting with the professor despite working right before the meeting.

Attempts to resolve:

- **Compared with previous code (No change in code was made prior to the issue occurring)**

- **Added print statements for pmpClass.py to check if data is being passed. (No issue)**
- **Added print statements for function define_target to check if data is being passed. (No issue)**
- **Added print statements for function set_motors_for2joints to check if data is being passed (No issue)**
- **Resetting the supervisor of the robot results in the define_target function being unable to find the target.**
- **Adding a new target to the world results in the same result as above.**

Overall Progress:
Issue is not resolved; robots are unable to move at all.

Next Week Objectives:

Date/Week: **Week 24 (10/10/2023)**

Objective 1: Have Separate Controllers for the four robots
Objective 1: Figure out the issue of how to represent the robot in rtb

Discussion /Lessons Learnt:
10/10/2023 Meeting
Professor had a look at the design of the 3DOF Robot, He pointed out that the shape acting as baselink was not connected to the Hingejoint, as the robot should ideally be a continuous chain of files connected in Webots. (There should be base link and the child of that link should be a joint, then the child of the joint should be a link, this will then repeat as you add joints to the robot).

Professor also suggested two solutions to resolve the issue of representation of the robot in RTB.
Solution 1: Use a program that can visually represent the robot you have created in the RTB.(The code in myBot.py). Which should be able to recreate a URDF file that

Solution 2: Use the RTB library to import it the URDF file

He also suggested to make separate code specific to the robot (Basically making different separate controllers for each). And focus on getting the Pmp to work on Four robots (These being the 2-joints planar robot, 3-joints planar robot, 3 Dof robot and a general robot.

Overall Progress:
Created separate controller folders for the Four robots, each folder has the PmpMain(The driver file), pmpClass(Is the same across all the robots), mybot(rtb representation of the robot), ikpy(handles ikpy urdf chain).

Tried solution 1, using swift to represent robot visually, able to create session but robot not displayed

Due to this attempted to redesign robot but current versions still not working

Next Week Objectives:

Date/Week: Week 23 (3/10/2023)

Objective 1: Figure out how to retrieve a 3D robot

Notes: Either redesign a custom one or import one into Webots

Discussion /Lessons Learnt:

3/10/2023 Meeting

Robot presented during meeting had major desgin flaw, was told to revise it completely or search online for URDF files of existing robots to import into Webots using Webots2Cyberatomics

URE3 Robot Testing:

Robot Name: URE3

Notes: URE3 has 6 joints named "shoulder_pan_joint," "shoulder_lift_joint," "elbow_joint," "wrist_1_joint," "wrist_2_joint," and "wrist_3_joint."

Status:

- Able to be imported to Webots.
- Moves with Pmp.
- Successfully moved with Pmp.
- Capable of following a target.
- Source: https://github.com/cyberbotics/webots/blob/released/projects/robots/universal_robots/protos/UR3e.proto

MyArm Robot Testing:

Robot Name: MyArm

Notes: MyArm is a custom robot with 3 joints named "joint1," "joint2," and "joint3."

Status:

- Successfully imported to Webots.
- Moves with Pmp.
- Capable of tracking a target.
- Note: This robot is custom-made.

Overall Progress:

Tested Pmp with the following robots
Able to import and test with URE3 and
able to move but need further modification to code base as this robot has 6 joints, which is more joints than the robots previously tested.

Created a custom robot named MyArm to test pmp implementation, seems to not have the poor design of the previous custom robots from last week, but still cannot reach target. Final Joint appears to move away from target. This robot was created based on guidance from this video, hence design is heavily reliant on theories and principles from this video (https://www.youtube.com/watch?app=desktop&v=m2SX3ubZrXs)

Regarding changes to Code:
MyBot.py
Modified function create_robot3dof_for3D to match the URDF file of the MyArm

PmpMain.py
Wrote a function (set_motors_forURE3) in PmpMain.py to set the retrieve and set the motors for URE3 Robot.

Next Week Objectives:

Date/Week: Week 22 (26/9/2023)

Objective 1: Create a proper 3D robot

Discussion /Lessons Learnt:
Previously created robot was simply a planar robot

Overall Progress:
Tested 2 joint in Y-axis is working, 3-Joints seems to be able to rotate along Z-axis but model seems to be breaking, this is most likely a result of the way the model is being designed.

Next Week Objectives:

Date/Week: Week 21 (19/9/2023)

Objective 1: Implement 3D Motion
Notes: Figure out how to make it come to a solution in a nice way
Objective 2: Go through Python Software Development Guidelines and create unit test
Objective 3: Modify rmse and time to be inputted from outside the Pmp class

Discussion /Lessons Learnt:
Ask to go through Python Software Development Guidelines. Following Python Style Guidelines (PEP Standards). Then design some unit test cases (based on guidelines from python docs).

X-Axis (Red Axis)
Y-Axis (Green Axis):
Z-Axis (Blue Axis):

Overall Progress:
3Joints robot seems to work (Apparently needed time to reach target), but movement does not look visually efficient.
Tested current version of the class from 2 up to 4 joint robots, seems to not generate errors.

Modified rmse and time to be inputted from outside the Pmp class

Unit Tests currently unable to receive k_ext in step and calculate_force function and experiences type error with rmse

Testing 3D on 2joint robot, robot is moving out of the stage and away from target but rotation along another axis seem possible. Modified joint motor rotation (orignial z=1, rest is 0)

Regarding changes to Code:

PMP class

Modified function execute_pmp_kinematics to now return self.q_traj, time, rmse, x_error

Modified x_target in function pmp_kinematics to no longer use 0 as axis but Index[2] of target positon

Modified funtion run_step to now use rmse > self.rmse_threshold and time <self.time_limit in the loop

Modified function step to now use self.step_size in self.q = self.q + (self.q_dot * self.step_size), this determines the size of the step the robot would move

Added global variables: self.rmse_threshold,self.time_limit,self.step_size

Notes: User must now declare these variables when using the class (initialization the class)

Next Week Objectives:

Date/Week: Week 20 (14/9/2023)

Objective 1: Rewrite PMP class so that it able to be configured for different jointed robots outside of the class

Discussion /Lessons Learnt:

Overall Progress:

Implemented method to accept different array sizes (meaning different joint number robots can be use without further modification to the main PMP class).

Regarding changes to Code:

PMP class

Modified self.q_current in function def to be set according to joints of robots

Modified q_traj in function run_step to be set according to joints of robots

Added the following as Global Variables:

- self.x=0
- self.q=my_bot.q
- self.jacobian = None
- self.force = None
- self.torque = None
- self.q_dot = None
- self.x_dot= None

Testing on 2 joints robot seems to be fine, but 3 joints robot although not passing an error, Ikresults seem to way off during personal testing. Calculations may need further adjustments

Next Week Objectives:

Date/Week: Week 19 (5/9/2023)

Objective 1: Rewrite Step Function to take and output x,x_dot,force,q,q_dot,torque

Objective 2: Move out functions that rely on the simulation, functions should be indedenpoent of librarys

Discussion /Lessons Learnt:

Regarding 3Joints, strange test results maybe a result of the current K_ext and a_int parameters being set, will require further development on the code to resolve this. Also results seem more reasonable during current date than previous tests last week.

Regarding the torso
Refer to Page 8 of Fyp article

Overall Progress:

The Tests seem to result in the arm almost reaching the target, not as close as previous tests before rewriting the step function.
Errors due to incorrect passing of arguments (K_ext) seem to require attention.

Moved the function that sets the joint position to the main driver file, the current implemenation seems to be not the same as previous tests.

Next Week Objectives:

Date/Week: Week 18 (29/8/2023)

Objective 1: Separate the Current Class file into several files to handle specific tasks
Objective 2: Implement and test on a 3-joints robot
Objective 3: Refractor code after Objective 1

Discussion /Lessons Learnt:

Re-evaluate the arguments being passed in the functions, as some appear to be redundant.
The Force and Torque values being passed especially need attention as errors during test typically are related to them.

Overall Progress:

Objective 1: Code still works on 2-joints robot after separation.

Changed the parameter needed for the driver file and the pmp class that deals with the supervisor. Other than this and adding import lines, no major changes were made to the existing code.

Objective 2: Created 3-joints robot and currently testing on separate world.

Robot appears to work with some modification to the existing code, mainly ensuring that an array corresponding to the number of joints is initialized.

Currently when setting the third joint to a translation of 1 along the x-axis seems to result in the calculation experiencing several issues, results var from stopping close to the target, to moving in the opposite target. Modifying it to have a translation of 0.5 seems to be fine so far in testing.

Notes: the two robots each are being tested with different controller files, each of them has their own version of the PMP class and associated files. Currently they are only minor differences between the two.

There is also a 3-Joints robot in the world used to test the 2-joints robot, if testing using both robots in the same world is needed.

Objective 3: Currently modifying the step function into separate functions

Notes: test conducted with 2-joints robot.

Test Results: returned ikresult calculations are different from the calculations generated with the previous code.

Next Week Objectives:

Date/Week: **Week 17 (22/8/2023)**

Objective 1: Rewrite Current PMP function into a Class

Objective 2: Implement Time Variance into the main PMP code

Discussion /Lessons Learnt:

In our recent meeting, we discussed incorporating variables within the class itself and introducing two functions: 'fkin' for forward kinematics and 'jacobian' for Jacobian matrix calculation. These functions will take 'q_current' as an input to represent the robot's current configuration. Notably, while located within the 'pmp' class, these functions won't form part of the core algorithm, ensuring a modular structure that separates kinematics from the algorithm itself.

Overall Progress:

- Removed several lines of code from the main PMP algorithm and separated them into functions that are called in the main PMP algorithm (Labelled as “step”), this was done to ensure any changes to libraries or toolboxes used would be done outside the main algorithm.
    - **PMP Class:** A new class PMP is defined, encapsulating the logic of the Position-based Motion Planning controller.
    - **Class Initialization:** The `__init__` method initializes the class attributes. It takes parameters such as `my_bot` (robot instance), `x_target`, `q_current`, `robotics_Library`, `K_ext`, `A_int`, and TBG for external control parameters.
    - **Stiffness and Admittance Settings:** `set_stiffness` and `set_admittance` methods allow setting the stiffness and admittance matrices for control.
    - **Kinematics Calculation:** The `calculate_fkine` method calculates forward kinematics based on the provided robotics library (either 'rtb' or 'ikpy'). If 'ikpy' is chosen, it loads a URDF file and calculates the transformation matrix.

        - Currently facing issues reading the URDF file
    - **Vector Calculations:** Methods like `calculate_error_vector`, `calculate_setpoint_x`, `calculate_current_x`, and `jacobian` handle vector calculations, such as error computation, setpoint calculation, current position calculation, and Jacobian computation.
    - **Torque and Velocity Calculations:** `calculate_Torque`, `calculate_q_dot`, and `calculate_Force` methods calculate torque, joint velocities, and control forces using provided matrices and vectors.
    - **Time-Based Gain (TBG) Calculation:** `calculate_tbg` computes a time-based gain factor based on the distance between the current and target positions.
        - The "**calculate_tbg**" function uses the "**x_error**" vector, which shows how much the robot's end effector position differs from the desired target. The function calculates the Time-Based Gain (TBG) that affects how the robot moves. It measures the distance between the current end effector position and the target. The TBG value is made by adding 1 to this distance, creating a factor that grows when the robot is far from the target. This makes the robot move more carefully as it gets closer to the target, ensuring smooth and controlled motion.
    - **Motion Planning Algorithm:** The `step` method implements the main position-based motion planning algorithm, updating joint positions and calculating errors iteratively until convergence.
    - **PMP Kinematics Function:** `pmp_kinematics` method takes a target name and calculates the joint trajectory using the `step` method.
    - **Motor Control:** `set_motors` method sets motor velocities and positions based on the calculated joint angles.
    - **Execution of PMP:** The `execute_pmp_kinematics` method encapsulates the execution of the PMP kinematics. It calculates the trajectory, final joint angles, and sets motor positions accordingly.
    - **Main Execution:** In the main part of the script, the Supervisor is initialized, and stiffness and admittance matrices are defined. An instance of the PMP class is created, and the PMP controller is executed for a specified target ('Ball').

- Currently Testing the Time Variance function
    - Currently seems to be working, as there is a slight decrease in time required to reach target. Recorded two of the experiments.

Next Week Objectives:

Date/Week: **Week 16 (12/8/2023)**

Objective 1: Read Autonomous Robots Article

Objective 2: Revise understanding PMP class code

Discussion /Lessons Learnt:

During meeting, lecturer instructed to reread the paper that code was based on and go through code to revise how it works.

Overall Progress:

Wrote a summary of the paper

Added comments to code

Next Week Objectives:

Date/Week: **Week 15 (16/5/2023)**

Objective 1: Go through Robotics Videos by ThatsEngineering

Objective 2: Go through iCub FW Kinematics Documentation

Objective 3: Implement RoboticsToolbox into iCub model

Discussion /Lessons Learnt:

Overall Progress:

Next Week Objectives:

Date/Week: **Week 14 (17/4/2023)**

Objective 1: Finish Report

Objective 2: Prepare slides for Presentation

Progress: Waiting for Feedback

Objective 3: Implement RoboticsToolbox into irb Model

Progress: getting importing error in environment

Discussion /Lessons Learnt:

Overall Progress:

Next Week Objectives:

Date/Week: **Week 13 (11/4/2023)**

Objective 1: Using IK to move Arm

Progress: Arm seems able to move but unable to reach target

Notes: The joint positions are currently be set with position over the limit of the joints, Try using x,y,z

Objective 2: RoboticsToolbox

Progress: Arm seems able to move but unable to reach target

Notes: The joint positions are currently be set with position over the limit of the joints

Objective 3: Set a new initial position for the model

Progress: Currently set Second Hinge joint parameters (right to 1.5 while left to –1.5) to a T-post position

Issue: Arm seems able to move but unable to reach target

Progress:

Notes: The joint positions are currently be set with position over the limit of the joints

Discussion /Lessons Learnt:

Overall Progress:

Next Week Objectives:

Date/Week: **Week 12 (3/4/2023)**

Objective 1: Using IK to move Arm

Progress: Arm seems able to move but unable to reach target

Notes: The joint positions are currently be set with position over the limit of the joints

Discussion /Lessons Learnt:

Robotics Toolbox is a MATLAB toolbox designed for robotics applications. It contains a set of tools and functions for simulation, analysis, and control of robotic systems. It can be very useful for solving inverse kinematics problems, which involve determining the joint angles required to reach a specific position and orientation.

Here are some ways that Robotic Toolbox can be useful with inverse kinematics:

- Robotic Toolbox provides a wide range of functions and algorithms for solving inverse kinematics problems, including analytical and numerical methods. These functions can be used to quickly and easily calculate the required joint angles for a given end-effector position and orientation.
- Robotic Toolbox allows you to define and simulate complex robot models with multiple degrees of freedom. This can be useful for testing and verifying your inverse kinematics solutions on different robot configurations and environments.
- Robotic Toolbox also includes functions for generating trajectories and motion planning, which can be used to generate smooth and efficient paths for the robot to follow. This can be useful for optimizing the robot's movement and reducing errors in the inverse kinematics solution.
- Additionally, Robotic Toolbox provides visualization tools that can help you visualize and debug your robot models and inverse kinematics solutions. This can be useful for understanding and improving the performance of your robotic system.

In summary, Robotic Toolbox can be a valuable tool for solving inverse kinematics problems and optimizing robotic systems. It provides a wide range of functions and tools for simulation, analysis, and control of robotics applications, which can help you quickly and easily solve complex problems and improve the performance of your system.

Overall Progress:

Next Week Objectives:

Date/Week: Week 11 (27/3/2023)

Objective 1: Read terms

Irb.webt look at inverse_kinematics and go through code

Use it to build the structure of the algorithm

- Read through the paper (PMP model)
- Read through implementation of PMP model (Both Student and supervisors)
- Look at icub Urdf/ check which joints are to be disabled in the masking (print(thearmlinks))
- Get an arm to move (ikpy/pmp)

Look at pmp

Get the arm move()

Objective 2: Create Environment objects (Table, items to grasp)

Progress: Done

Discussion /Lessons Learnt:

Explanation of Kinematic Terms

Inverse Kinematics is the most important concept being implemented in the Project. To explain briefly, it refers to the computation of the joint angles that are needed to move a ***limb*** (in the case of this project) to reach the desired end-effector position (***Target***). Using the results of these calculations to set new positions and rotate the joints to angles that bring the limb to the target. This method is commonly done using a Jacobian matrix.

Forward Kinematics, in simple terms, refers to computing the end point of the limb based on the positions of its joints; it involves using transformation matrices to calculate how each joint moves the end point and combining them together to get the final position and orientation of the end point.

The **Jacobian matrix** is a mathematical tool that describes the relationship between the motion of a robot end-effector and the movement of individual joints. By utilizing the Jacobian matrix, it becomes possible to calculate the necessary motion of each joint required to achieve a desired velocity and position for the end-effector. Once the required joint velocities are determined through the Jacobian matrix, they can then be integrated over time to obtain the joint angles necessary for the robot to reach the desired end-effector position.

Stiffness is related to the condition number of the Jacobian matrix. This being a measure of how sensitive the output of a function is to small changes made into the input.

A high condition number indicates that the system is stiff, this means that any small errors in the velocity of the desired end-effector (target) can result in large and prominent errors in the joint velocities. This can cause the system to become unstable or unable to accurately achieve the desired trajectory.

Meanwhile, a low condition number indicates that the system is compliant, meaning that it can tolerate and handle small errors in the velocity of the desired end-effector (target) without causing large errors in the joint velocities. This can make the system more stable and allow for more flexibility in the trajectory. An ideal state to maintain in a system.

Admittance is the method of controlling a robot by specifying the desired behaviour of the force and velocity of the end-effector (target). Then by adjusting the admittance of the system until achieving the desired level of stiffness or compliance required or specified for development and testing.

The Equilibrium Point Hypothesis (EPH) proposes that the posture of the body is not directly controlled by the brain but is instead a biomechanical consequence of equilibrium among these forces. This means that complex actions can be achieved without a complex optimization process, by allowing the intrinsic dynamics of the neuromuscular system to seek its equilibrium state when triggered by intended goals.

The **Passive Motion Paradigm** (PMP) builds upon the Equilibrium Point Hypothesis (EPH) to explain how the brain controls movement. According to PMP, the brain uses an internal simulation process on the body schema to determine how to move the joints to achieve a specific goal.

Overall, PMP suggests that the brain controls movement by simulating the distribution of forces across the joints, rather than by directly controlling the movement of each joint. This approach is more efficient and requires less computational power than other methods, such as **inverse kinematics.**

Overall Progress:

Next Week Objectives:

Date/Week: Week 10 (21/3/2023)

Objective 1: Redesign the move function that uses these limbs as an argument based on the code shown below

Progress: Working

Code outlined during meeting:

```
    Def MoveLimb(leftarm=None, rightarm=None, neckNone,...){
        If left_arm:
            For joint, values in left_arm:
                    Joint.setPosition(values['position'])
                  Joint.setVeclocity(values['velocity'])
```

To configure the left arm to a specific position using joint values or velocities, you can create a copy of the left arm named **"left_arm_next"** and fill it with the desired joint values or velocities. This approach can be applied to any body part. To move the limb to the configured position, you can use the MoveLimb function and pass the "**left_arm_next**" and "**neck_next"** as Arguments

Objective 2: PMP

Progress: To Be Discussed Further

Additional Notes

PMP: First we need to write an algorithm that takes in a goal position [x,y,z] generates joint angles left arm [myjointvalueset,] joint velset

Objective 3: Create Environment objects (Table, items to grasp)
Progress: Not Started

Read terms
Irb.webt look at inverse_kinematics and go through code
Use it to build the structure of the algorithm

- Read through the paper (PMP model)
- Read through implementation of PMP model (Both Student and supervisors)
- Look at icub Urdf/ check which joints are to be disabled in the masking (print(thearmlinks))
- Get an arm to move (ikpy/pmp)

Discussion /Lessons Learnt:
Regarding the controller
In the section #Get Joints
Here list of joints is initialized for each body part (referred to as "**limbs**" in the project) of the robot.

In the Section *#Defines the MoveLimb function*
The Function takes in the limbs of the robot as arguments, being able to take up to six possible arguments.

- Then it would move the joints of the limbs that were passed to the function.
- The limb argument passed to the function would enter a for loop that would iterate over each joint in the limb (here in the form of a list)
- Then the position and velocity of each joint would be set based on values specified in *values* dictionary
- The *position* value determines the desired position of the joint,
- while the *velocity* value determines the desired velocity at which the joint should move to the desired position.

In the Section *#movesets*
Here you would select a move set for a limb by creating lists of tuples.
Each tuple contains a joint from the corresponding limb list and a dictionary with the joint's desired position and velocity.

- These tuples are then passed as arguments to the *move_Limb* function which moves the robot's limbs according to the specified positions and velocities.

Overall Progress:
Move Function can now set position and velocity for joints (Currently testing with left and right arm)

Next Week Objectives:

Date/Week: **Week 9 (13/3/2023)**

Objective 1: Organize the limbs of the Icub into a dictionary structure
Progress: may need refinement

Objective 2: Create a move function that uses these limbs as an arguments
Issue#1: When setting position for limbs, the error "AttributeError: 'dict' object has no attribute 'setPosition'" occurs.
Progress: In Progress

Objective 3: Configure a specific position for the left arm
Progress: In Progress

Discussion /Lessons Learnt:
Regarding the framework of the functions for icub

- To create a framework for the functions for the icub model, it is suggested by the supervisor converting the limbs to a dictionary structure.
- Allowing the ability to Call the move function with the dictionary of limbs as arguments can then be implemented.

Organizing the limbs into this dictionary structure would make it easier to access and manipulate each limb individually.
However, it is important to make sure that the dictionary keys correspond to the actual limb names and that the values contain the necessary information such as position and velocity.
Using Objective 1 as an example, the AttributeError mentioned above indicates that there may be an issue with how the limbs are currently being accessed and manipulated.

Overall Progress:
Currently limbs have been converted to the dictionary structure.
The Move function is still currently not working

Next Week Objectives:

Date/Week: **Week 8 (7/3/2023)**

Objective 1: Rename terms for the joints of the icub model
Progress: In Progress

Objective 2: Add Sensors to each joint
function1(getting access to the joins/Senors)
function2(read sensors to get current value)
Issue#1: trouble understanding the syntax of the position sensor class in Webots.

Progress: In Progress

Objective 3: call and send four arguments to initialize
function3(initalize)
By default, it would set in place the idle position of the model (initialize everything)
it would decide which limb to initialize
function(arg1,arg2,arg3,arg4)

With Arguments including the Neck, torso,right hand and left hand

Progress: In Progress

Objective 4: Create move function
Using the body parts of the icub model as the arguments
Decide on the Format on how to present the output (eg, list,dict)

body_part(as an arg) = 7 joints contain in the arms (others may vary)
When moving the arm (such as in the forward position, then you would get the joint angles), list them down to create the dict, so later you can issue a command to move the model to a set position.

Progress: In Progress

Objective 5: Hand Implantation
The Following Options

1. do the conversion of the existing icub model
2. add hand joints yourself (Including the fingers)
3. replace with a gripper

Progress: On Hold

Discussion /Lessons Learnt:
Sensors added to checks if correct or incorrect movement has occurred and obstacle detection.
Regarding the legs:
The Following options were discussed during the meeting

- freeze the legs and stick legs to the ground
- Make the robot sit down
- Create a chair for icub to sit on (The most supported option)
-

View softillusion webots tutorials
Tutorial 13 for matters regarding model hand
Tutorial 16 for matters regarding the joint sensors

Overall Progress:
Added Sensors to each joint
Added information regarding the neck joint to the table of joints

Next Week Objectives:

Date/Week: **Week 7 (27/2/2023)**

Objective 1: Create a function that takes in input (7 Joint Values) which would set the joints to that value

Progress: In Progress

Issue#1: Unable to call the joint (Error: Device 'name' was unable to be found')

Status: Appears to be resolved

Solution: DEF name was not declared

Notes: Currently applying function to arm joints only,

Code is heavily based on the code use for the Ned Sample provided in Webots.

Issue#2: Arm movement will cause entire model to collapse

Current Theory: Reduce either position of joint or limit the velocity of the moment

Status: Appears to be resolved

Notes: Setting velocity below 2 seems to negate issues regarding collapsing model. Currently proceeding with velocity set to 1.

Objective 2: Figure out the range of the joints and List them for all the joints.

Progress: Done

Notes: *Further addition and refinement to the table will be very likely and needed*, as joints for the hand (Palm and Fingers) have yet to be implemented. Not to mention other values that may come up later.

Objective 3: Resolve the conversion Issue

Option 1: ROS package

https://github.com/robotology/icub-models#use-the-models-from-python-helper-library

ROS acts as a server for robot commands

Each joints receives Input from these servers

Issue: Currently in Webots, there is no connection to the ROS server

Option 2: Create a new model

Progress: On Hold

Discussion /Lessons Learnt:

Regarding the hand (*icub_stand.wbt*) may require motor addition if Objective 3 is not resolved

Regarding the Table of Joints

27 Joints are currently recorded in the table list of joints. With the *maximum position, minimum position, maximum velocity and maximum torque* of each Indvidual joint recorded.

- **Maximum Torque** referring to the largest possible force needed to rotate the joint
- **Maximum Position** referring to the largest possible position the joint can rotate to
- **Minimum Position** referring to the smallest possible position the joint can rotate to
- **Maximum Velocity** referring to the fastest possible value it can move the joins to
-

Regarding the Joint Function

The reference file is labelled as "*ned_python.py*" and can be found in the sample worlds

Under the Device folder of the joint, *need to declare the DEF name* (This will be the name used in the function call), Please do this for all joints in the robot.

Each arm currently has 7 Joints.

Each joint should have velocity set to 1

Refer to the Table of Joints when setting things such as position, velocity and torque.

Not doing so will result in the errors.

Overall Progress:

Next Week Objectives:

Date/Week: Week 6 (24/2/2023)

Objective 1: Importing icub_stand.wbt

Progress: Resolved

Issue#1: The EXTERNPROTO declaration is not working error is still not resolved.

Action Taken: The path specified in the declaration leads to an empty folder with no assets, currently looking for said assets

Action Taken: Attempted to download missing proto files and place them into path specified. Downloaded from https://raw.githubusercontent.com/cyberbotics/webots/R2022a/projects/objects/backgrounds/protos/TexturedBackground.proto

Solution: Replaced EXTERNPROTO declartion using the declarations called by the icub_stand.wbt pre-installed

Objective 2: Matters regarding icub model.

Progress: In Progress

Note: Attempted to Convert existing icub_stand.wbt model into proto to resolve issues in Objective 1

Issue#1: Currently unable to convert Urdf files into proto using previously installed tool, Results in the command returning that it is unable to find the file.

Discussion /Lessons Learnt:

Invalid pathing specified will result in multiple issues.

The Conversion process between URDF to Proto may need to revise.

Overall Progress:

Current Issue regarding the conversion process from URDF to Proto

Next Week Objectives:

Date/Week: Week 5 (14/2/2023)

Objective 1: Importing icub_stand.wbt

Progress: In Progress
Issue#1: Unable to import, Multiple errors regarding not being able to detect necessary files for world. Console Prompt showing and asking “to Ensure correct world is being opened”
Current Theory: The EXTERNPROTO declaration is not working, as objects are present, but textures are missing.
Issue#2: Unable to create new protos files under current project.

Objective 2: Investigating issue with icub model.
Progress: In Progress

Discussion /Lessons Learnt:
Discussed during meeting with Supervisor regarding importing issues with icub models, three solutions were suggested to resolve this issue.
Option 1: Resolve the invalid escape character error, in summary continue to use the current model.
Option 2: Open an issue on github regarding icub as a proto
Option 3: Use the Icub_stand model and add features from the current model.

Icub_stand model can be found on
https://raw.githubusercontent.com/cyberbotics/webots/master/projects/robots/robotcub/icub/worlds/icub_stand.wbt

Issue regarding icub as a proto can be found on
https://github.com/cyberbotics/webots/issues/5345

Overall Progress:
The issue regarding import models has still not been resolved

Next Week Objectives:

Date/Week: **Week 4 (6/2/2023)**

Objective 1: Getting motor for icub robot to work (Experiment with the movement)
Progress: In Progress
Issue #1: Robot falls down the floor of the world simulation when simulation is started
Theory: Due to Issues of Importing

Objective 2: Import icub robot into Webots
Progress: In Progress
Issue #1: Unable to **Convert Urdf to Proto (Resolved)**
Solution: Use Cd command to locate path of model file
Issue #2: able to import icub robot into Webots but unable to observe said robot (**iCubGenova09**)
Theory: Appearance/ Texture packages maybe be undetected despite pathing to the texture files seems to be correct.

Objective 3: Review ProtoFile issues/Ned
Progress: Done

Discussion /Lessons Learnt:
Downloaded Icub Model from https://github.com/robotology/icub-models and extracted to a file simply named “File” and placed in the desktop.

Search for Cyberbotics/URDF2Webots on Github (https://github.com/cyberbotics/urdf2webots)
Used `pip install --no-cache-dir --upgrade urdf2webots` to install the utility tool needed to convert Urdf model file to proto.

Used `Python -m urdf2webots.importer --input=”model.urdf” --output=”Output_File”` command to convert the model file.
Then moved the converted file into the protos file. After that, when in Webots import the robot from the current project.

Note for Obj2 (Packages maybe required for some models)

Overall Progress:
Urdf file can be converted into proto, then imported into Webots, but appearance and overall stability of the model are unable to imported correctly

Next Week Objectives:

Date/Week: **Week 3, Date 31-1-2023**

Objective 1: Refine knowledge regarding Controllers.
Progress: Done

Objective 2: Understand Proto-files.
Progress:
Having issues importing Proto Nodes (asset no cached)
Resolved but New node creation failed (expected node or Proto Name)

Objective 3: Understand how joints work in Ned.
Progress: in Progress

Discussion /Lessons Learnt:
Create new (File > New>New Proto)
Proto files allow robot nodes to imported
Controllers(Set the Physical Steps, Max Speed, initialize robot node, get then set motors followed by their speed as 0)

Overall Progress: (summaries any findings; any achievements, challenges, difficulties encountered)
Ctrl + Alt, to apply force.

DEF field defines objects.
Proto-Files define the robot, such as the links and range of the joints.
Proto-Files would contain the ProtoName, ProtoField(Defines the modifiable fields of Proto Node) and the ProtoBody(Defines Root Node).

Next Week Objectives:
Import icub robot

Date/Week: Week 2, Date 29-1-2023

Objective 1: Create a simple controller program
Progress: Done

Objective 2: Import icub robot
Progress: in Progress

Discussion /Lessons Learnt:
Declare bound objects before messing with physics field.

Overall Progress: (summaries any findings; any achievements, challenges, difficulties encountered)
Robot can navigate pass simple obstacles using the controller program.
Able to add spherical objects.
Having trouble understanding DEF-USE mechanism.

Next Week Objectives:
Import icub robot
Understanding DEF-USE mechanism more clearly.

Week 1 (17/1/2023)

Date/Week: Week 1, Date 17-01-2023

Objective 1: Install Webots, Mini Conda and Python and Update packages
Progress:
Issues running webots (No QT Plugin Error),
Current Theory regarding Cause: Unsupported Windows Version,
Conclusion: Theory was correct, and everything is working now

Download links
https://docs.conda.io/en/latest/miniconda.html
https://cyberbotics.com/

Objective 2: Create GitHub account and make a repository
Progress: Done and Available on https://github.com/MorganChew/18b9062-Final-Year-Project

Objective 3: Add Objects in Webots
Progress: Done

Objective 4: Create a simple controller program
Progress: Still in Progress

Objective 5: Import iCub robot
Progress: Not yet Started

Discussion /Lessons Learnt:
Check for software version requirements

Overall Progress: (summaries any findings; any achievements, challenges, difficulties encountered)

Webots, Mini coda and Python Installation were successful.
GitHub repository has been made and presently has one commit.
Objects can be added in Webots.

Next Week Objectives:
Finish simple controller program
Import icub robot