Implementation of a Reinforcement Learning system on a micro-controller-based module for stabilizing an actuated platform

PROBLEM STATEMENT 1 - Team ID: IXT101026(Sumukh, Nimisha, Vishnu)

Abstract of our Approach :

We were provided with a collection of code-files containing a simulation of a single axis rotation motor platform built using - the *main* file, *model* file, *trajectory* file, *header* file and the RL (Reinforcement learning) file. The RL file is indicative of the stability logic implemented using the concept of PD controllers. We were told to develop code that replaced the PD Controller logic with a reinforcement learning system by defining rewards to rectify errors arising due to lack of stability from the files mentioned above.

As a side note, the approach presented in this document does not involve the use of Tensorflow Lite or any of its sub-packages due to issues pertaining to compatibility of code with STM32F103 microcontroller.

The Reinforcement Learning logic/concept has been written in C++ using Object Oriented Programming. The concept of training the model using rewards has been incorporated in a similar manner as well.The code in this document ,by itself, gives the desired result.We have derived mathematical relations considering a sinusoidal trajectory and have used the derived relations to compute the error between the desired and obtained values

The desired value of torque and obtained value of torque are attempted to be brought close to each other and the same is reflected in the output screenshots and screen recording.

The following document entails the detailed approach and the working of the solution to the aforementioned problem statement.

The Mathematical Approach to Deriving Expressions :

➔ **The following variables have been considered:**

H(x)	=	the desired torque,
Y	=	obtained value of torque,
Theta	=	obtained value of angular displacement of single axis rotation motor,
Theta dot	=	obtained value of derivative of theta which is angular velocity

➔ **The following constants have been used:**

- From header file
    1. *pi=3.14159265358979323846264338327950*
    2. *2*(pi)=6.2831853071795864769252867665590*
    3. *dt=0.001 seconds*
- From model file
    1. *Inertia=0.1*
    2. *Initial theta= 0.5 radians*
    3. *Initial theta dot=0.1 rad/sec*
- From trajectory file
    1. *omega = 0.1*
    2. *amplitude = 1*

➔ **The following equations for torque and desired torque have been derived from the equations in given files**.

Y (obtained torque) = Inertia/2dt {(theta/dot) +theta dot}

H(x)(desired torque)=

*Inertia*amplitude/2dt((sin(angle)/dt)+omega(cos(angle))*

The derivation for the above two equations is as follows :

FROM MODEL.C

the = the_dot x dt

the_dot = accel x dt

$\therefore$ the = accel x$(dT)^2$

accel= torque/inertia

$$\therefore \text{the} = \frac{\text{torque}}{\text{inertia}} \times dt^2$$

$$\therefore \text{torque} = \frac{\text{the} \times \text{inertia}}{dt^2} \quad - \text{①}$$

Similarly,

$$\text{torque} = \frac{\text{the\_dot} \times \text{inertia}}{dt} \quad - \text{②}$$

① + ② $\Rightarrow$

$$\text{torque} = \frac{\text{inertia}}{2dt}\left(\frac{\text{the}}{dt} + \text{the\_dot}\right) - \text{③}$$

FROM TRAJECTORY FILE:-

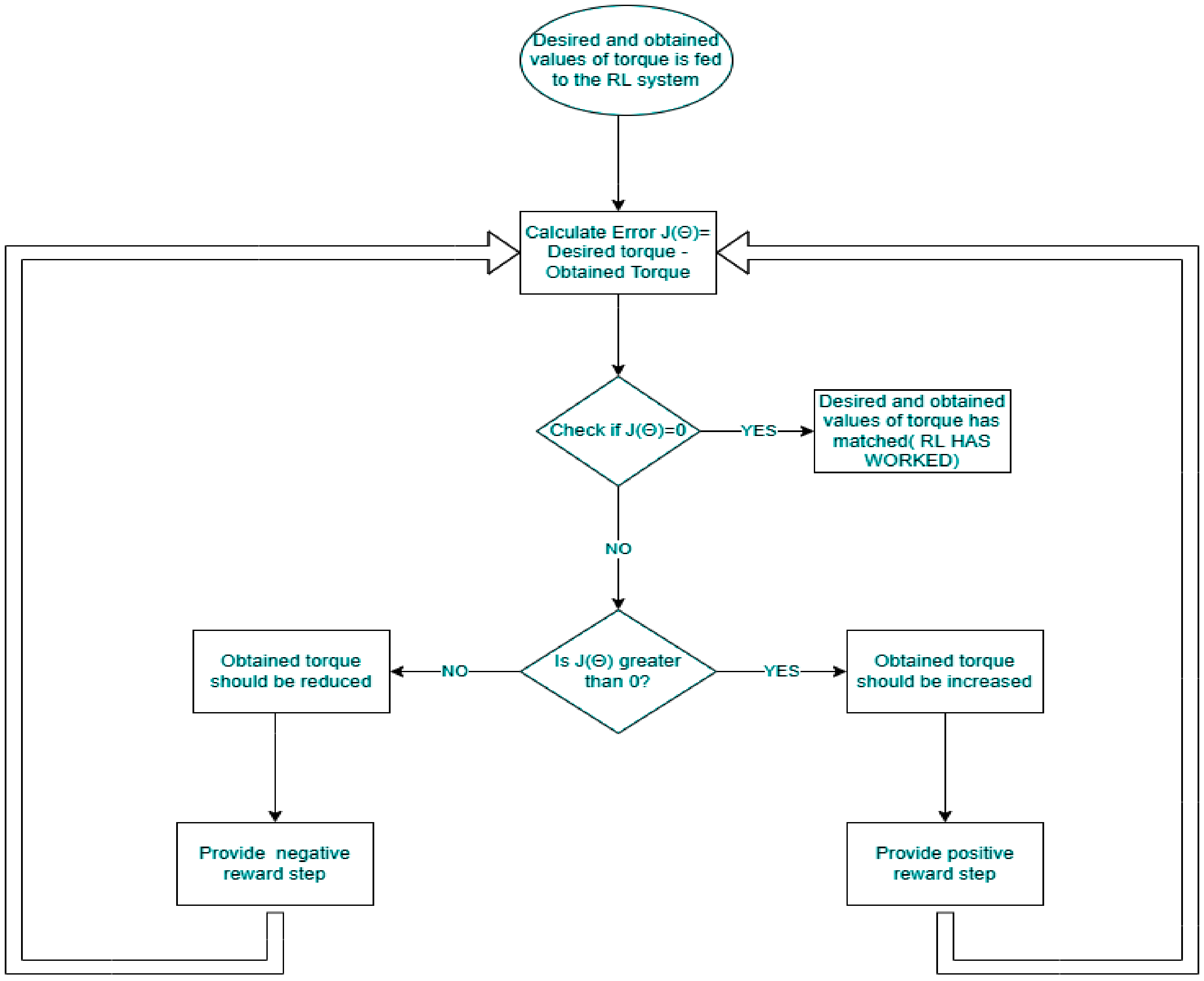
des_the = amp * sine(angle)

des_the_dot = amp * omega * cos(angle)

$$\therefore \text{des\_torque} = \frac{\text{amp} * \text{inertia}}{2dt}\left[\frac{\text{sine(angle)}}{dt} + \text{omega} * \cos(\text{amgle})\right]$$

- ➢ The torque linearly depends on theta and theta dot. From the above derived equations,we can state that:
- ➢ If obtained values of theta and theta dot approach the desired values of theta and theta dot then we can infer that the obtained torque will approach the desired value of the torque
- ➢ Torque is the amount of rotational force we need to apply on the actuated platform to stabilize it if it has been destabilized .In our approach, we have used Reinforcement Learning to bring torque closer to desired torque to achieve stabilization.

<u>RL logic Flowchart:</u>


Desired and obtained values of torque is fed to the RL system
Calculate Error J(Θ)= Desired torque - Obtained Torque
Check if J(Θ)=0
YES
Desired and obtained values of torque has matched(RL HAS WORKED)
NO
Is J(Θ) greater than 0?
NO
Obtained torque should be reduced
YES
Obtained torque should be increased
Provide negative reward step
Provide positive reward step

C++ CODE:

```
#include<bits/stdc++.h>
#include<iostream>
#include<vector>
#define PI 3.14159265
using namespace std;

class Torque {
    public:
    float initTorque;
    Torque(float x) {
        initTorque= x;
    }
    float dt = 0.001;
     float amp = 1;
     float omega = 0.1;

    const float m_inertia = 0.1;

    float computey(float m_inertia, float dt, float theta, float
thetaDot) {
    static float omega = 0.1F;
     static float amplitude = 1;
     static float angle = 0;
    float dt1 = 0.001;
     angle += omega * dt1;
    float theta1 = sin (angle*PI/180);
    float thetaDot1 = cos(angle*PI/180);
    float y = (m_inertia/(2*dt))*((theta1/dt) + (thetaDot1));
        return y;
    }
   float sinetheta = -0.05; //theta
   float cosinetheta = 0; //thetadot
float computeTorque(float m_inertia, float dt, int amp,float
sinetheta, float cosinetheta) {
    static float omega = 0.1F;
     static float amplitude = 1;
     static float angle = 0;
    float dt1 = 0.001;
     angle += omega * dt;
    float sinetheta1 = sin (angle*PI/180);
    float cosinetheta1 = cos(angle*PI/180);
       float h_x = (m_inertia/2*dt)*(amp*sinetheta1/dt + 
amp*omega*cosinetheta1);
    //    cout<<"Testing output of h_x: "<<endl;
       return h_x;
    }

float computeError(float m_inertia, float dt, float theta, float 
thetaDot,int amp,float  sinetheta, float cosinetheta) {


float y = (m_inertia/2*dt)*((theta/dt) + thetaDot);
   static float omega = 0.1F;
     static float amplitude = 1;
     static float angle = 0;
    float dt1 = 0.001;
     angle += omega * dt;
    float sinetheta1 = sin(angle*PI/180);
    float cosinetheta1 = cos(angle*PI/180);
        float h_x = (m_inertia/2*dt)*(amp*sinetheta/dt + 
amp*omega*cosinetheta);
      // return h_x;
    float j_theta = h_x - y;
    return j_theta;
    }


int rewardStep;

    float rewardFunc(float j_theta, float rewardStep) {
```

```
    while(1) {
     static float omega = 0.1F;
     static float amplitude = 1;
     static float angle = 0;
    float dt1 = 0.001;
     angle += omega * dt1;
    float theta1 = sin (angle*PI/180);
    float thetaDot1 = cos(angle*PI/180);
        float y = (m_inertia/(2*dt))*((theta1/dt) +
(thetaDot1));
         if(y>0) {
            y-=rewardStep;
        }else if(y<0) {
            y+=rewardStep;
        }else {
            cout<<"IDEAL VALUE REACHED..."<<endl;
        }

        return y;
    }
    }

    float updatedComputeTorque(float j_theta) {

        while(1) {
    float y =  computey(0.1, 0.001,0.05,0.1);
    float h_x= computeTorque(0.1,0.001,1,-0.05,0.0);
        if(h_x==y) {
            cout<<"MATCH..."<<endl;
        }else if(h_x!=y) {
            computeError(2.0,2.0,0.02,0.01,1,-0.05,0.0);
        }
  return (h_x-y);
    }
    }
```

```
};

//driver
int main() {

    Torque t1(0.0);
    cout<<"Process started"<<endl;
    cout<<"linking objects at runtime"<<endl;
    cout<<endl;

    while(1) {


  float resy = t1.computey(0.1, 0.001,0.000005,0.0000001);
  cout<<endl;
  cout<<"COMPUTING DESIRED TORQUE..."<<endl;
  cout<<resy<<endl;
  cout<<endl;

  float resce =  t1.computeError(2.0,2.0,2.0,3.0,1,-0.05,0.0);
  cout<<"COMPUTING ERROR..."<<endl;
  cout<<resce<<endl;
  cout<<endl;


float resuct =  t1.updatedComputeTorque(0.001);
     cout<<"COMPUTING UPDATED TORQUE..."<<endl;
  cout<<resuct<<endl;
  cout<<endl;

cout<<"----------------------------------------------------------------
----------------------------------------------------------------
-"<<endl;
    }

        float rew1 =  t1.rewardFunc(1.0,0.1);
```

```
      cout<<" Reward..."<<endl;
    cout<<rew1<<endl;
    cout<<endl;

      return 0;

 }
```

Explanation of code:

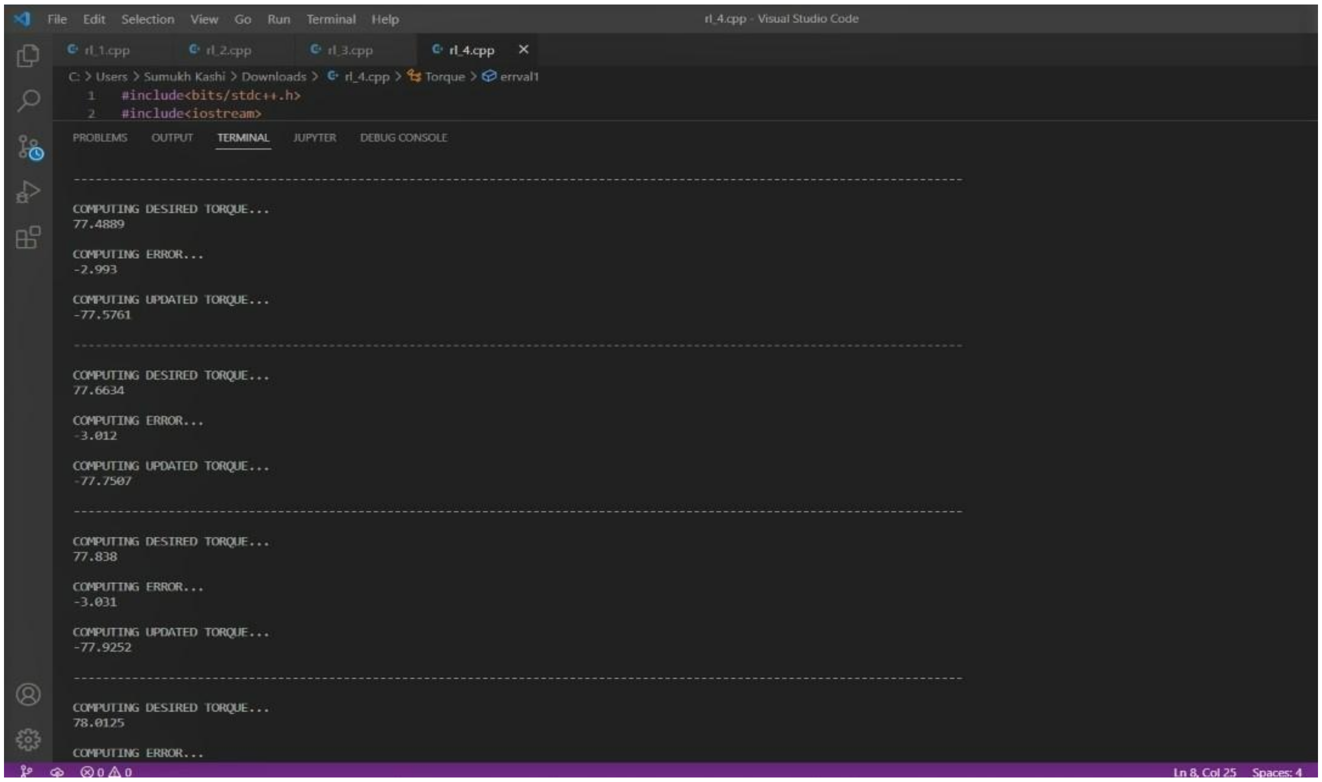
- Following a few imports that support the use of various math functions and classes, a state space has been defined that serves the purpose of housing the body of the code.
- A class called torque has been defined wherein initial values of parameters like dt, amplitude, omega and inertia are declared.
- Additionally, a constructor to initialize the torque object has been defined using a single parameter.
- The body of the Torque class comprises 5 functions each responsible for computing the value of a single component, the aggregation of which will lead to reinforcement of torque values in a positive direction thereby achieving stabilization.
- The first function (*computey*) in the said class is responsible for computing torque.
- The equation for computing this parameter has been elaborated in the section that deals with the mathematical derivation of parameters.
- Plugging in values of theta and theta dot from sine and cosine waves respectively, the value of torque (here *y*) has been calculated.
- The second function (computeTorque) accomplishes the task of computing the value of desired torque, the equation for which has been described in the section that deals with mathematical derivation of parameters.
- An identical snippet of code draws values from sine and cosine waves to calculate the value of desired torque.

- The third function (*computeError*) is responsible for computing the difference between values of torque and desired torque and returning the difference subsequently.
- A dynamic programming approach has been used to plug in values returned by the two functions described above and compute the difference between the two. A variable called j_theta has been used to store the result of this said computation.
- A function (*rewardFunc*) to compute rewards is defined. A scheduler has been placed to check if the error factor tends to zero on every iteration and a value (*rewardStep*) is used to adjust values of the error when it does not align with the ideal reference value.
- The last function (*updatedComputeTorque*) in this class is responsible for computing the updated value of torque. The updated value of torque depends on the value of error that is generated in the previous function.
- A driver function has been defined outside the Torque class to call all functions defined in the preceding section.
- An object of class Torque has been instantiated and initialized using the pre-defined constructor.
- To ensure that values are generated dynamically, a single infinite while loop has been wrapped around all functions. The object has been used to call the functions defined, accordingly.
- The output has been formatted to conform with the readability factor.

This summarizes the working of the reinforcement learning system employed to achieve stability of the actuated platform.

<u>Final Result and Conclusions:</u>

As mentioned before we have aimed to show the desired and obtained values of torque to approach each other as the iterations progress and the difference between to progressively decrease which can be observed in the below screenshot


File Edit Selection View Go Run Terminal Help
rl_4.cpp - Visual Studio Code
rl_1.cpp
rl_2.cpp
rl_3.cpp
rl_4.cpp
C: > Users > Sumukh Kashi > Downloads > rl_4.cpp > Torque > errval1
PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
Ln 8, Col 25 Spaces: 4

```
#include<bits/stdc++.h>
#include<iostream>
```

```
------------------------------------------------------------------------------------------------------------------------

COMPUTING DESIRED TORQUE...
77.4889

COMPUTING ERROR...
-2.993

COMPUTING UPDATED TORQUE...
-77.5761

------------------------------------------------------------------------------------------------------------------------

COMPUTING DESIRED TORQUE...
77.6634

COMPUTING ERROR...
-3.012

COMPUTING UPDATED TORQUE...
-77.7507

------------------------------------------------------------------------------------------------------------------------

COMPUTING DESIRED TORQUE...
77.838

COMPUTING ERROR...
-3.031

COMPUTING UPDATED TORQUE...
-77.9252

------------------------------------------------------------------------------------------------------------------------

COMPUTING DESIRED TORQUE...
78.0125

COMPUTING ERROR...
```

Observations of values of variables from screenshot (from a set of 3 iterations for example)

Desired Torque	Obtained /Updated Torque
1) 77.8380	77.9252
2) 77.6634	77.7507
3) 77.4889	77.5761

Hence, we can conclude that our Reinforcement learning logic can progressively decrease the deviation between the desired and obtained values of torque.

With respect to the microcontroller interfacing,the successfully executed *C++* code has been modified to be compatible with the environment in order to facilitate the compiling process.The environment used is the *Keil-uVison5 MDK-Arm* version development environment for *Cortex*.

With the hardware(*STM32F746-DISCO*) provided to us we interfaced the peripherals with the computer using a Mini USB Cable for debugging and dumping the code onto it.

Further developments:

If our RL algorithm, dumped on the microcontroller can be deployed on an actual experimental setup such as a single axis rotation motor , the performance of the code can be exhibited in a better way.

Since our access to hardware was only limited to the STM32F746-DISCO board provided, we haven't been able to demonstrate working on an actual experimental setup. However it can be deployed on a physical setup.