***ELECTRONICS DESIGN REPORT***

***M A R S***



Team division :

1. Control system
2. Communication
3. Power electronics

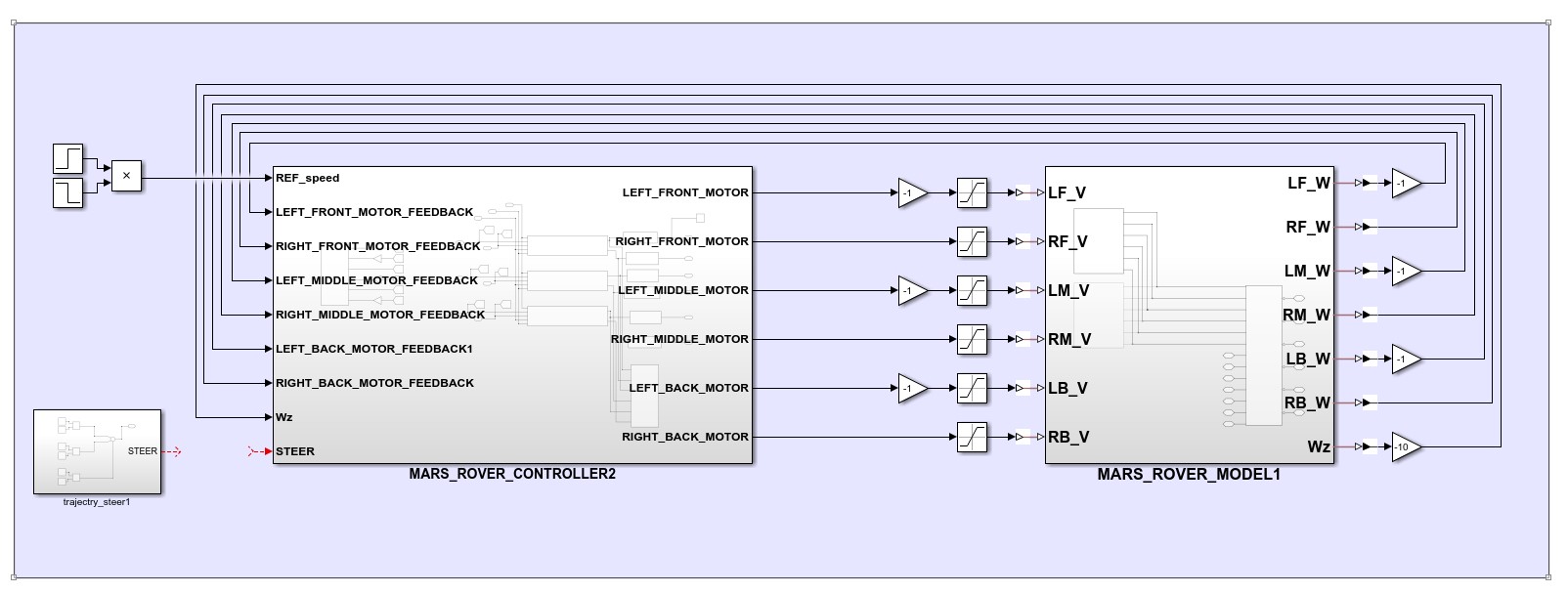
# CONTROL SYSTEM

* The aim of the team is to design and build controllers for the rover's robotic arm and mobility unit.
* The mobility controller and manipulator controller are the two main controllers we have to build.

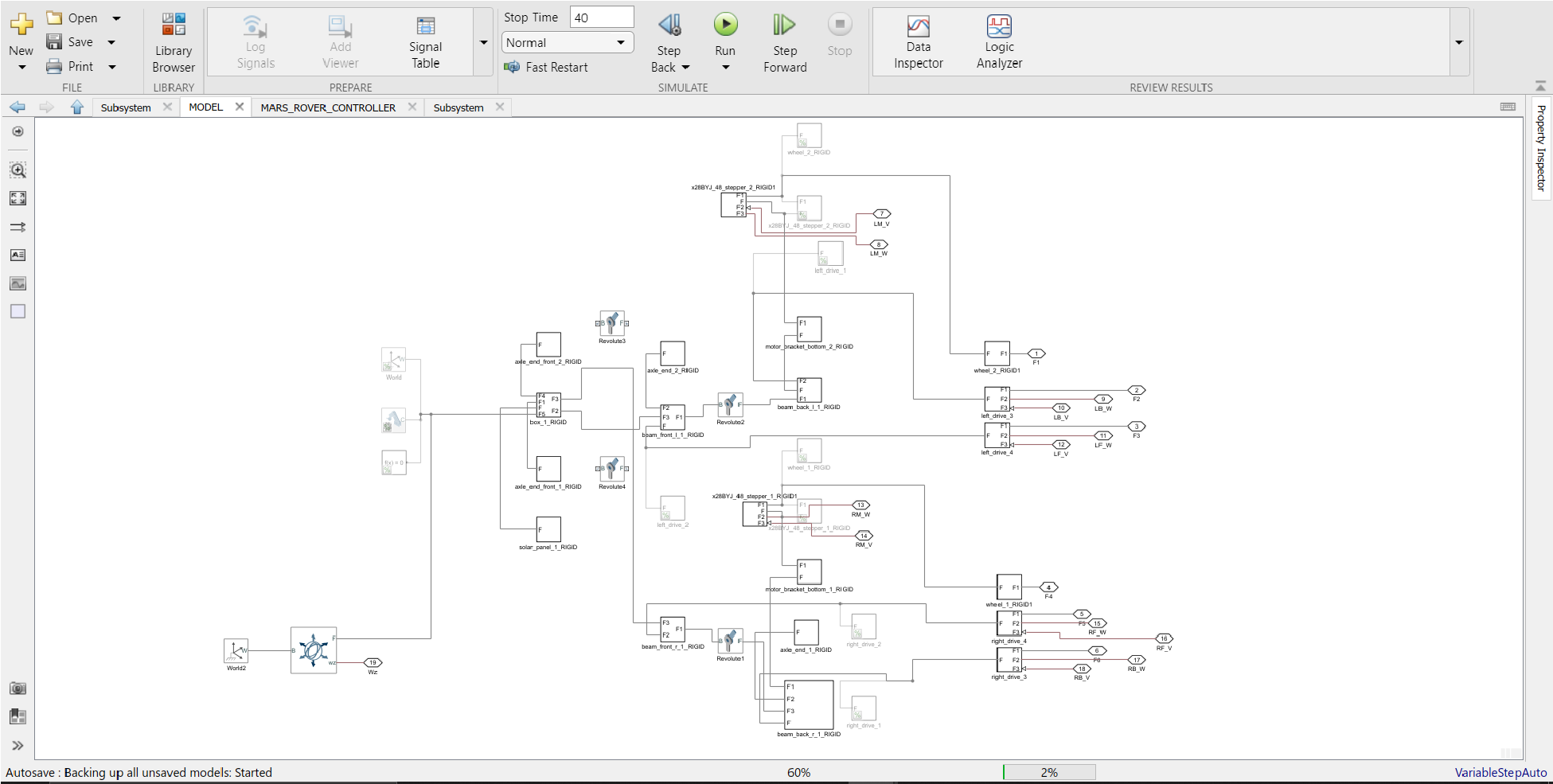
## 1) MOBILITY CONTROLLER

**DESIGN PHASE :**

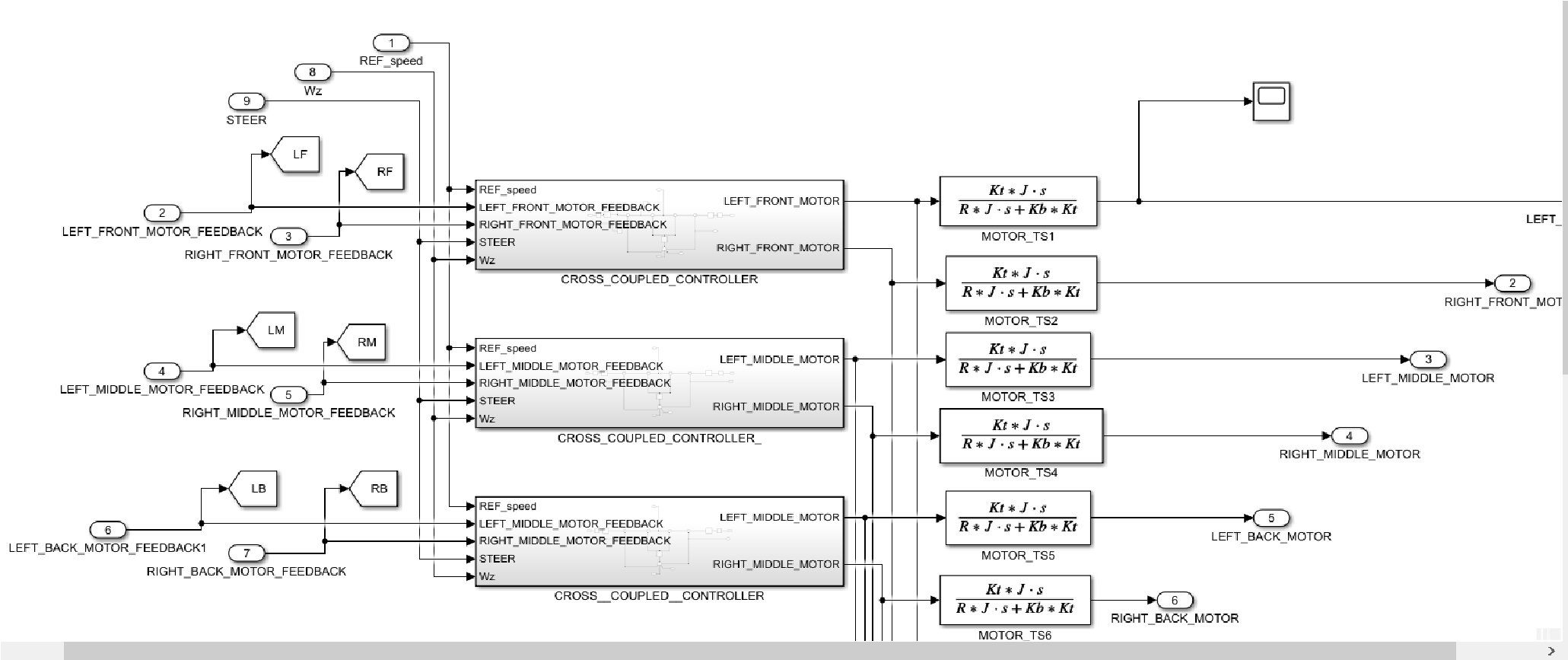
* We started from
* a requirements phase, from the requirements we started building a controller which satisfied the requirements .
* In the design phase we mainly stick to MATLAB for designing and testing the control designs .
* In matlab we started , building the controllers for dc motor speed control to the whole mobility controller .
* Here is the mobility controller model , where the controller will get the feedback from the rover model which is created using multibody .



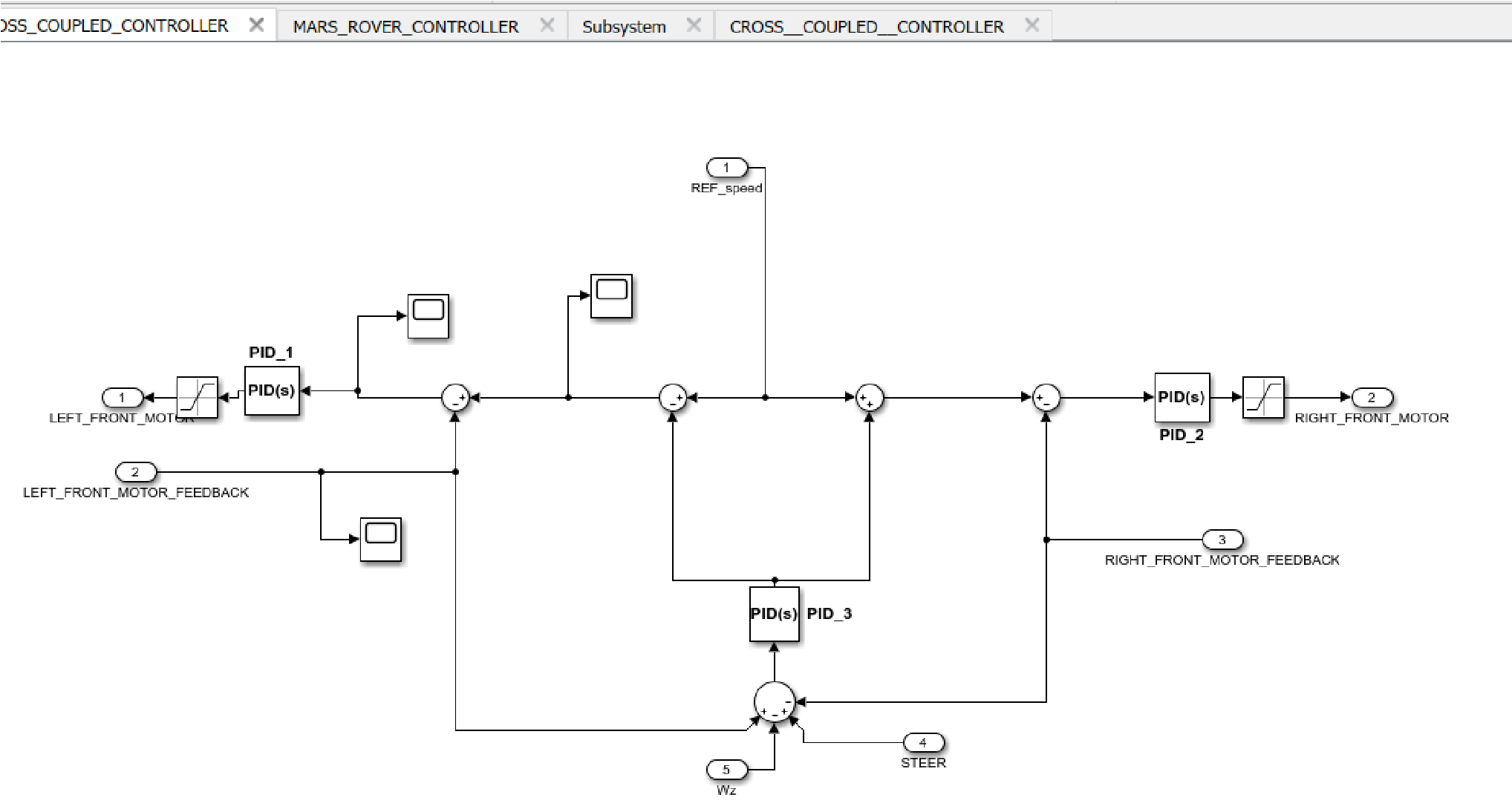
* In matlab , we created an environment using simulink multibody for testing the controller. ● We imported a rover design from onshape to matlab multibody to test the controller.



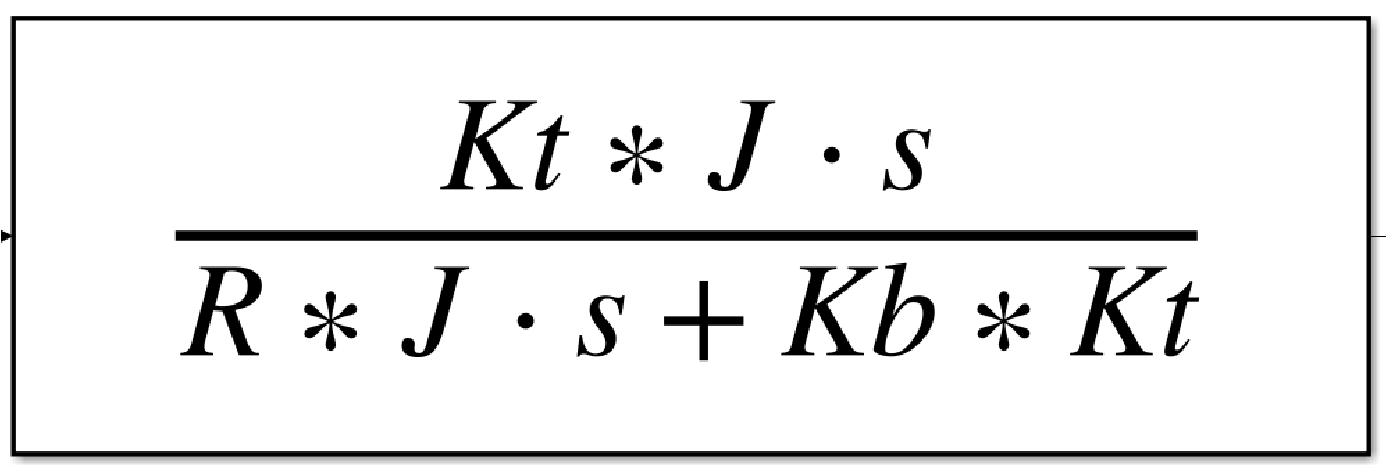
* This environment used a multibody multiphysics library used to create frictional forces .
* After the environment is created , we can start testing the controller , to control the rover in the environment .
* Our controller design embedded a cross coupled controller for more precise control .



* We designed a controller , which can control two motors, left and right , so these controllers can be used to control all 6 motors of the rover .



* Here in the cross coupled controller pid\_1&2 are responsible for controlling the speed of each motor.
* The pid\_3 is responsible for maintaining the relative speed between left and right motors . it is also responsible for eliminating the deflection of the rover from the straight line of motion
* Below is a transfer function of the motor , which gives the torque output from the input voltage, such that we can feed the torque as input to the environment model .



Where :

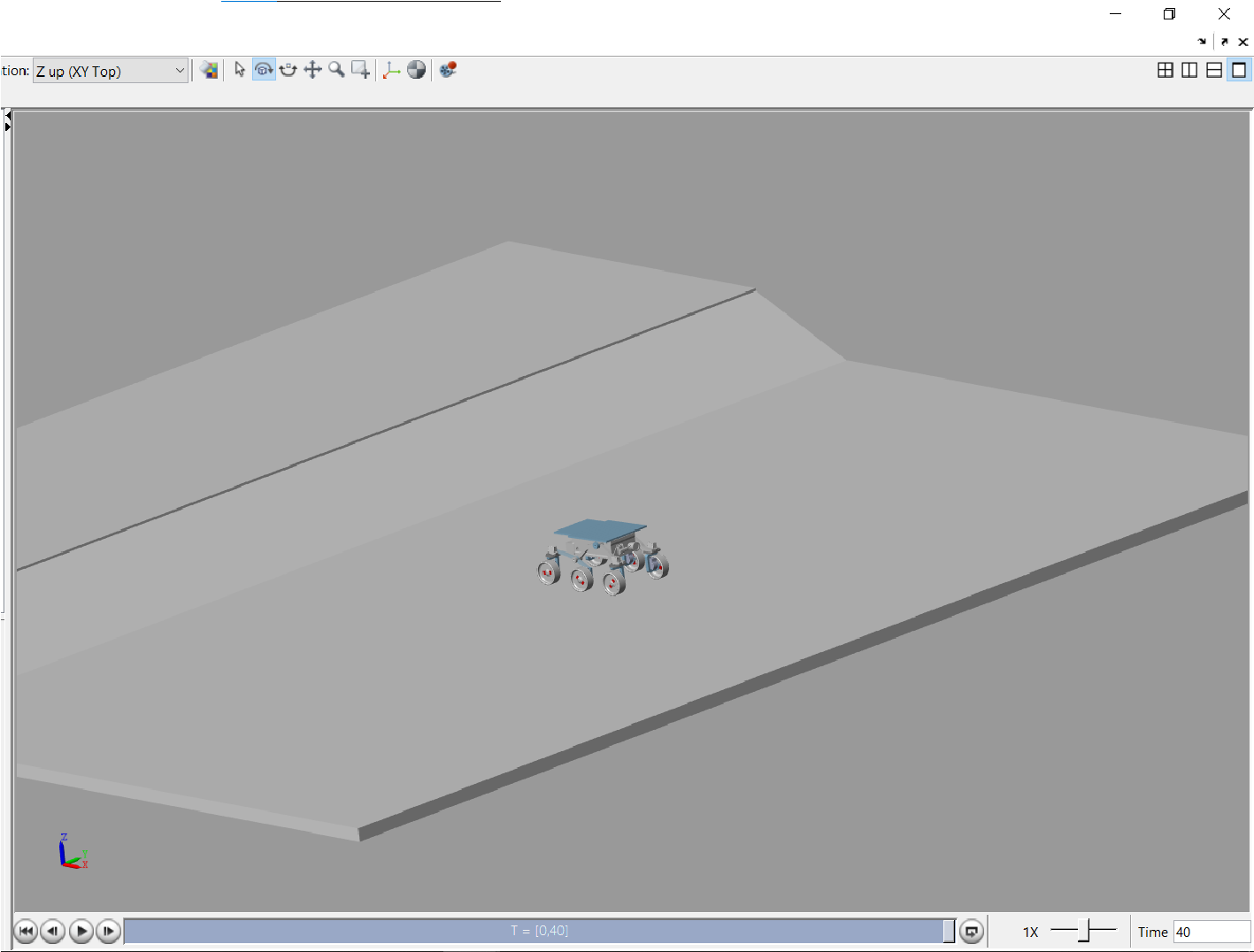
Kt = torque constant

J = moment of inertia

R = resistance of coil

Kb = back emf constant

* Here is the , final mechanical explorer view of the simulation of rover



You can find the simulation video in [MARS DRIVE UNDER](https://drive.google.com/drive/folders/1yWL2I2A6uEod_iDaPujqYcqBVu2hNAHq?usp=sharing) [ELECTRONICS](https://drive.google.com/drive/folders/1yWL2I2A6uEod_iDaPujqYcqBVu2hNAHq?usp=sharing)

* Finally we tested and verified the controller . so now we have to embed this design to the real hardware and software
* So , we planned to use a stm32f103c8 microcontroller for implementing each cross coupled controller , which will control two wheels in the rover . so , total 3 stm’s and a nano which is responsible for communicating with three stm’s and with the main processor and also provide imu data .

**IMPLEMENTATION PHASE :**

* In this phase , we are responsible for code the individual controllers so as to act like a controller that we designed .
* So , we splitted this phase into 3
  1. Fetching & decoding the dc motor encoder data, for feedback of speed and position .
  2. Coding the controller according to the design in matlab
  3. Enabling the i2c bus communication for getting commands and data from the master .

**ENCODERS DATA :**

* We are using a quadrature encoder for feedback , where we have to read two signals from the encoder .
* There will be a lag between two signals which is responsible for detecting the direction of the motor rotation .
* We are using a stm31 to read the four signals from two encoders .
* In stm , we are enabling external picchange intercepts , for reading the encoder data .
* Here is the arduino code for stm.

////////////////// READING ENCODER1 DATA USING THE SYSTICK TIMER(MILLIS) /////////////////////

void ENCODER1() { /// determining value of micros in the time of an interrupt, where micros won't update

current\_state1 = MICROS(); pulse\_time1 = current\_state1 - previous\_state1; previous\_state1 = current\_state1;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if((GPIOB\_BASE->IDR >> (uint32\_t)13) & 0x1) DIR1 = 1; // reading PB13 pin for second

encoder signal for direction

else DIR1 = -1;

F1 = DIR1\*(double(1000000/double(pulse\_time1\*N\*G))); E1 = E1 + DIR1;

}

* The above code will execute , when a rising edge is detected in the encoder signal .
* Inside the function , we were measuring the time in which interrupt occurred , and stored in current\_state\_1 ,
* Then , we determined the time period of the encoder pulse to calculate the frequency .
* Every time an edge is detected , the variable E1 increments by 1 .
* We were using a custom made function MICROS() , because if we use micros inbuilt function inside interrupt, NVIC creates a problem , where the millis counter won't update such that we will get the wrong time period .

/////////////////// MICROS FUNCTION WHICH WORKS INSIDE INTERRUPTS //////////////////// uint32\_t MICROS(void) {

uint8\_t int\_flag;

int32\_t systick\_counter,current\_state;

|  |
| --- |
| systick\_counter = SYSTICK\_BASE->CNT; |

// System timer(Systick) base

if(0b1 & SCB\_BASE->ICSR >>26){ //Check if Systick interrupt pending flag is set

int\_flag = 1;

|  |
| --- |
| systick\_counter = SYSTICK\_BASE->CNT; |

//Re-read Systick Timer

}

|  |
| --- |
| else int\_flag = 0; |

// Systick interrupt pending flag is not set

current\_state = (systick\_uptime\_millis \* 1000) + (SYSTICK\_RELOAD\_VAL + 1 systick\_counter) / CYCLES\_PER\_MICROSECOND;

if(int\_flag) current\_state += 1000;

return current\_state;

}

For more reference , to understand the above code refer link :

[STM32 for Arduino - Connecting an RC receiver should be easy, right?](https://www.youtube.com/watch?v=utTi0awlUzg&t=767s)

**CONTROLLER CODING IMPLEMENTATION :**

* We needed to code the controller design , so we coded the controller in arduino ide for stm32f103 .
* We splitted the coding part into two parts :

1) Coding the cross coupled motor controller .

* This code should be implemented in a stm32.
* This controller code is responsible for controlling the two left & right motors by coding the desired controller according to matlab .
* We were using PD as a base controller

2) Coding mobility master controller

* This master is responsible for running dead reckoning algorithms and fetching data from IMU and the main communication node .
* We will use an arduino nano for this master .
* This controller will fetch data from the main communication node via can protocol .

//-------------------------- CROSS COUPLED CONTROLLER ------------------------------//

// STEERING PID FOR FRONT MOTORS

ERROR\_S = (L\_M\_VEL - R\_M\_VEL) - (FEEDBACK\_L - FEEDBACK\_R) -

FEEDBACK\_STEER;

//----------------------------------------------------------------------------------------------------------------

S\_PID = PID(ERROR\_S, PREV\_ERROR\_S, &S\_I,S\_Kp, S\_Ki,S\_Kd, SAT\_S\_PID,

S\_PID,TIMER1); // PID FUNCTION USAGE

//----------------------------------------------------------------------------------------------------------------

TIMER1 = micros();

SAT\_S\_PID = SATURATE(S\_PID,-SAT\_PID\_VALUE,SAT\_PID\_VALUE); // SATURATE THE PID

PREV\_ERROR\_S = ERROR\_S;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* The above code is pid\_3 which is responsible for maintaining relative speed between the two motors
* This pid feedback also gets the imu data , to avoid the deflection . ● However, the pid\_3 controller is also a PI controller .

// PID CONTROLLER FOR LEFT FRONT MOTOR

ERROR\_L = L\_M\_VEL - FEEDBACK\_L - (S\_PID \* 0.3);

//------------------------------------------------------------------------------------------------------------L\_PID =

PID(ERROR\_L,PREV\_ERROR\_L,&L\_I,L\_Kp,L\_Ki,L\_Kd,SAT\_L\_PID,L\_PID,TIMER2); // PID

FUNCTION USAGE

//-------------------------------------------------------------------------------------------------------------

TIMER2 = micros();

SAT\_L\_PID = SATURATE(L\_PID,-SAT\_PID\_VALUE,SAT\_PID\_VALUE); // SATURATE THE PID

PREV\_ERROR\_L = ERROR\_L;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// PID CONTROLLER FOR FRONT RIGHT MOTOR

ERROR\_R = R\_M\_VEL - FEEDBACK\_R + (S\_PID \* 0.3);

//------------------------------------------------------------------------------------------------------------R\_PID =

PID(ERROR\_R,PREV\_ERROR\_R,&R\_I,R\_Kp,R\_Ki,R\_Kd,SAT\_R\_PID,R\_PID,TIMER3); // PID

FUNCTION USAGE

//-------------------------------------------------------------------------------------------------------------

TIMER3 = micros();

SAT\_R\_PID = SATURATE(R\_PID,-SAT\_PID\_VALUE,SAT\_PID\_VALUE); // SATURATE THE PID

PREV\_ERROR\_R = ERROR\_R;

}

* The above code is a PI controller , which is responsible for maintaining a desired set speed .
* We used a custom PID( ) function for implementing the above code .
* This pid function also eliminates the integral windup problem , and saturation problem .
* This controller gets the input data from the main communication node via I2C master arduino nano .

///////////////////////////////////////// PID FUNCTION /////////////////////////////////////

double PID(double ERROR\_, double PREV\_ERROR\_, double \*I,byte Kp,byte Ki,byte Kd,double SAT\_PID,double PID,uint32\_t TIMER){ // calculate the pid according to gains .

double P,D; double FREQ;

FREQ = micros() - TIMER;

P = ERROR\_ \* (double)Kp; // finding proportional \*I = \*I + ERROR\_ \* (Ki / FREQ); // finding integral

\*I = \*I + 2\*(SAT\_PID - PID); // eliminating the integral windup .

D = (PREV\_ERROR\_ - ERROR\_) \* (double)Kd \* FREQ; // finding derivative term

return (P + \*I + D);

}

**MAIN CONTROLLER CODE** :

● This main controller is responsible of doing the following tasks :

* + 1. Receiving data from the main communication node thusing can protocol and sending it to stm’s
    2. Dead reckoning
    3. Collecting data from IMU

1. Receiving data from main communication node
   * We will receive data from a MCP2515 can module ● We will communicate with spi with the module .
2. Dead reckoning
   * This code is responsible for determining the position of the rover with respect to a starting position .
   * It will use encoder data and imu data to determine its location

Below is the corresponding code for dead reckoning

/////////////// DETERMINING THE AMOUNT OF DISTANCE THE ROVER HAD MOVED /////////////// DELTA\_D = (DELTA1 + DELTA2 + DELTA3 + DELTA4 + DELTA5 + DELTA6 )/(6\*N\*G); // DETERMINING THE AVERAGE ENCODER COUNT ...

DELTA\_D = (DELTA\_D + (RPM1 + RPM2 + RPM3 + RPM4 + RPM5 + RPM6)/(F\*60\*6))/2; // APPROXIMATING ROVERS MOTION THROUGH VELOCITY DATA ...

|  |
| --- |
| DELTA\_D = DELTA\_D\*(2\*3.1415926\*WHEEL\_RADIUS); |

// UNITS IN "mm"

///////////////// DETERMINING THE COORDINATES ///////////////////////////////////////

X1 = X1 + DELTA\_D\*sin(YAW\*DEG\_RAD)\*cos(PITCH\*DEG\_RAD); // units in mm Y1 = Y1 + DELTA\_D\*cos(YAW\*DEG\_RAD)\*cos(PITCH\*DEG\_RAD); // units in mm

Here X1 and Y1 are the final coordinates of the rover .

3) Collecting data from IMU

* We were using an adafruit IMU for getting the whole yaw , pitch and roll orientation data of the rover .

**I2C COMMUNICATION :**

* Here , we are using 3 stm’s for controlling 6 motors , so there is a master controller which is responsible for running dead reckoning algorithms and fetching data from IMU and the main communication node .
* So, we are using I2C communication for sending data from master to the stm’s .
* In this communication the stm’s are coded as I2C slaves .
* In this communication , we were sending encoder data to master and receive , speed commands for two motos .

THESE ARE FEW DOCS WE PREPARED WHILE DESIGNING PHASE

LINKS :

1. [MARS CONTROLL DESIGNS](https://docs.google.com/document/d/1sgywngVuJXw6LpATxBV_xwJsSaNYNI6Jful30brdC58/edit?usp=sharing)
2. [MARS ROVER TEAM](https://docs.google.com/document/d/1oCEBGpo0fXKEY9Q3U21dOBlN5CxURw_D2ujpGSz6_uI/edit?usp=sharing)
3. [Control system design](https://docs.google.com/document/d/1F1W-Wdt9tYWzoilqf7zwNSRwcpmRwR-1U0gDbO8qr6Y/edit?usp=sharing)
4. [Power modules and devices](https://docs.google.com/document/d/1xlg9jss_BKxoFM7dz9JPoEVOO7rQ6Eu0JMB2i7112JE/edit?usp=sharing)
5. [MARS ROVER STEERING CONTROL DESIGN](https://docs.google.com/document/d/1y7k1E69MQaU3EH5Q9_15e_0ppx8j1t6KA1ZeGFNmmyo/edit?usp=sharing)
6. [SENSOR FUSION](https://docs.google.com/document/d/1XZswZraj4UVCmjmEYA5nIGhDNY9wFg_9UrDx5bY-4v4/edit?usp=sharing)
7. [Motors](https://docs.google.com/document/d/1eCIlsks6GiPGVva6KuqWUAuTb3r5ljimiZ3mKtvFoUI/edit?usp=sharing)

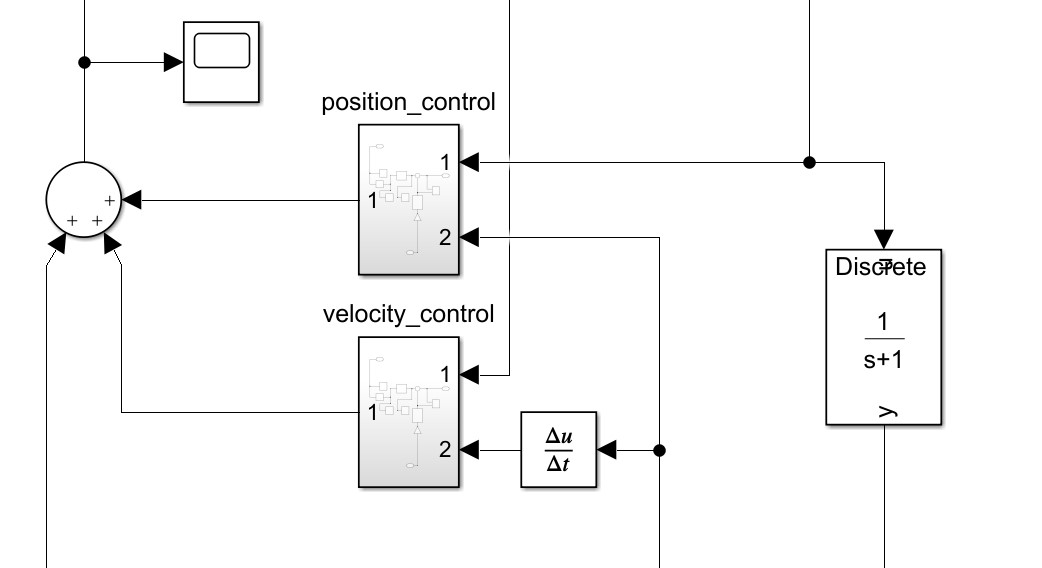
2) ***MANIPULATOR CONTROLLER***

* This manipulator controller's main aim is to position the manipulator joints to its appropriate location .
* So, this controller has two modes :
  1. Getting each joint angle from an external processor (ie.. getting the joint angle data from a rovers on board processor using inverse kinematic algorithm ).
  2. (Or) determining its own joint angles from 3 DOF inverse kinematic algorithms running inside the controller (This feature can be used when the processor algorithm fails ).

### Design phase

* In this phase , our main aim is to design a joint controller in matlab for a manipulator .
* Such that we created a position controller in matlab
* Our main control aim is to develop a current based position controller .
* According to that we developed a controller which controls speed and position

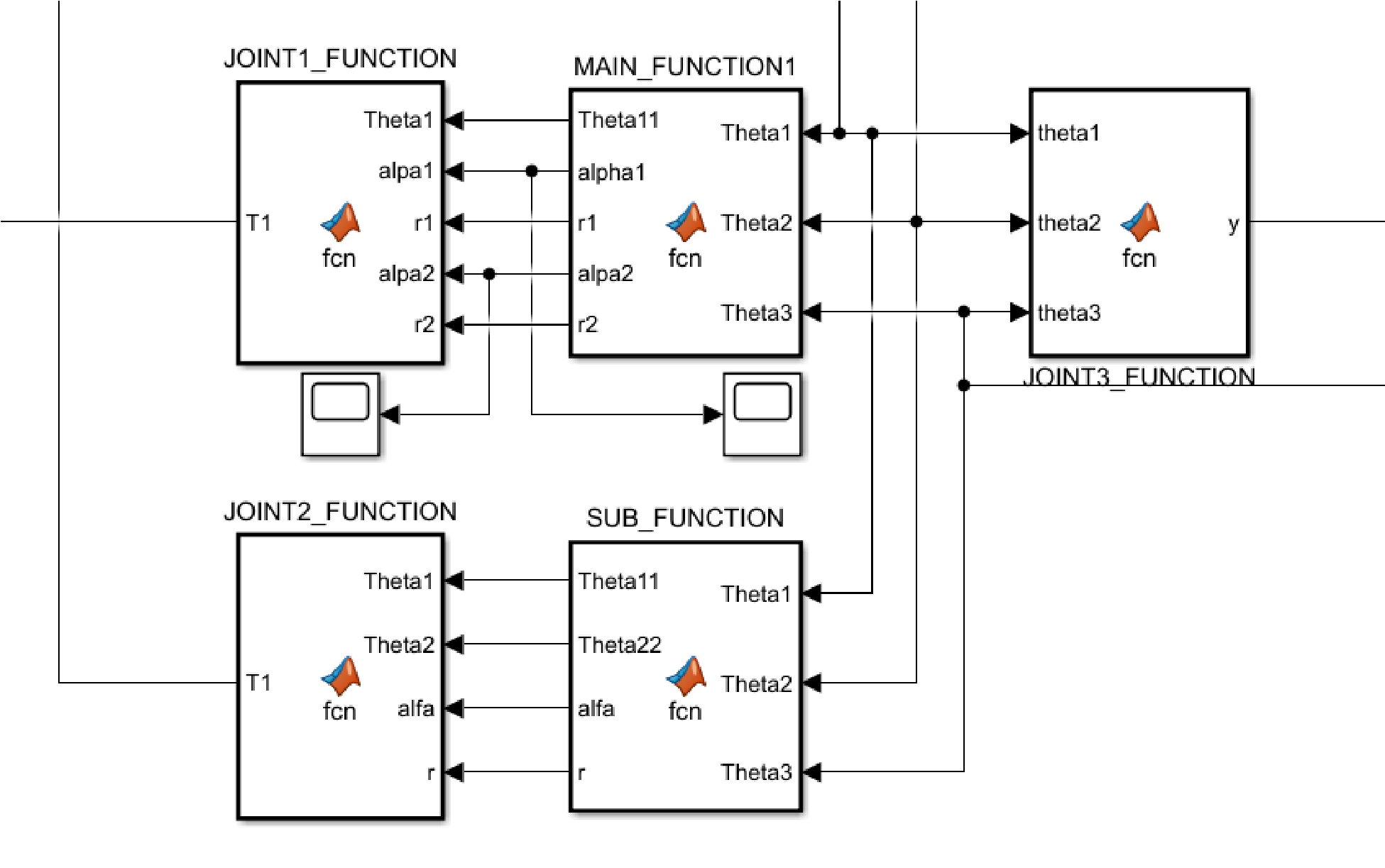
#### Controller



In the above , the results of the both controllers will together command the joint .

* The third sum is , feedforward path which will provide the appropriate torque to each joint to eliminate the effect due to gravity .

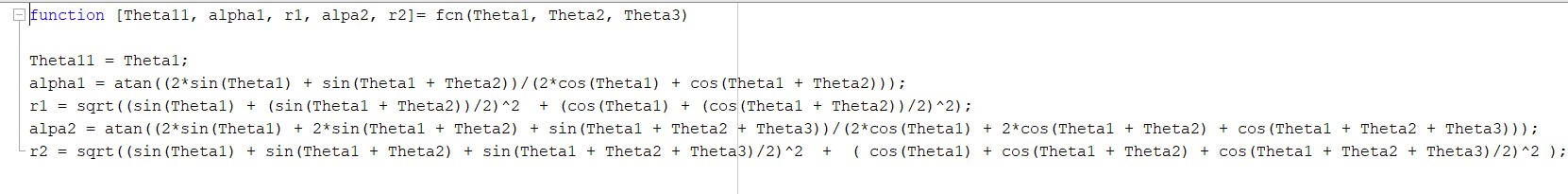
Feed forward path :



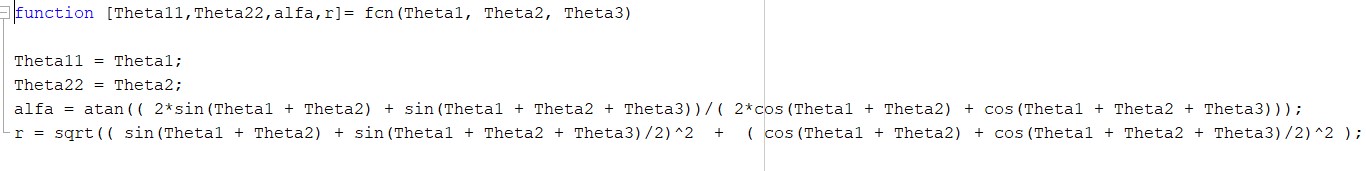
These are the matlab code for determining the gravitational torque acting on three joints

**MATLAB CODE :**

MAIN\_FUNCTION CODE :-

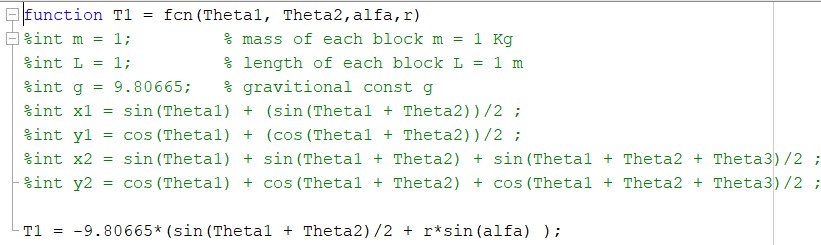
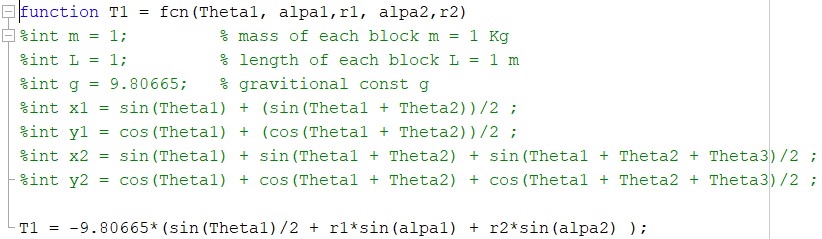


SUB\_FUNCTION code :



This function takes inputs as angles of three joints , and performs some tegromentric calculations to determine the effective center of mass of the whole three joints .

From that data , we can determine the torques , acting on each joint from manipulator parameters and the data from the main function .

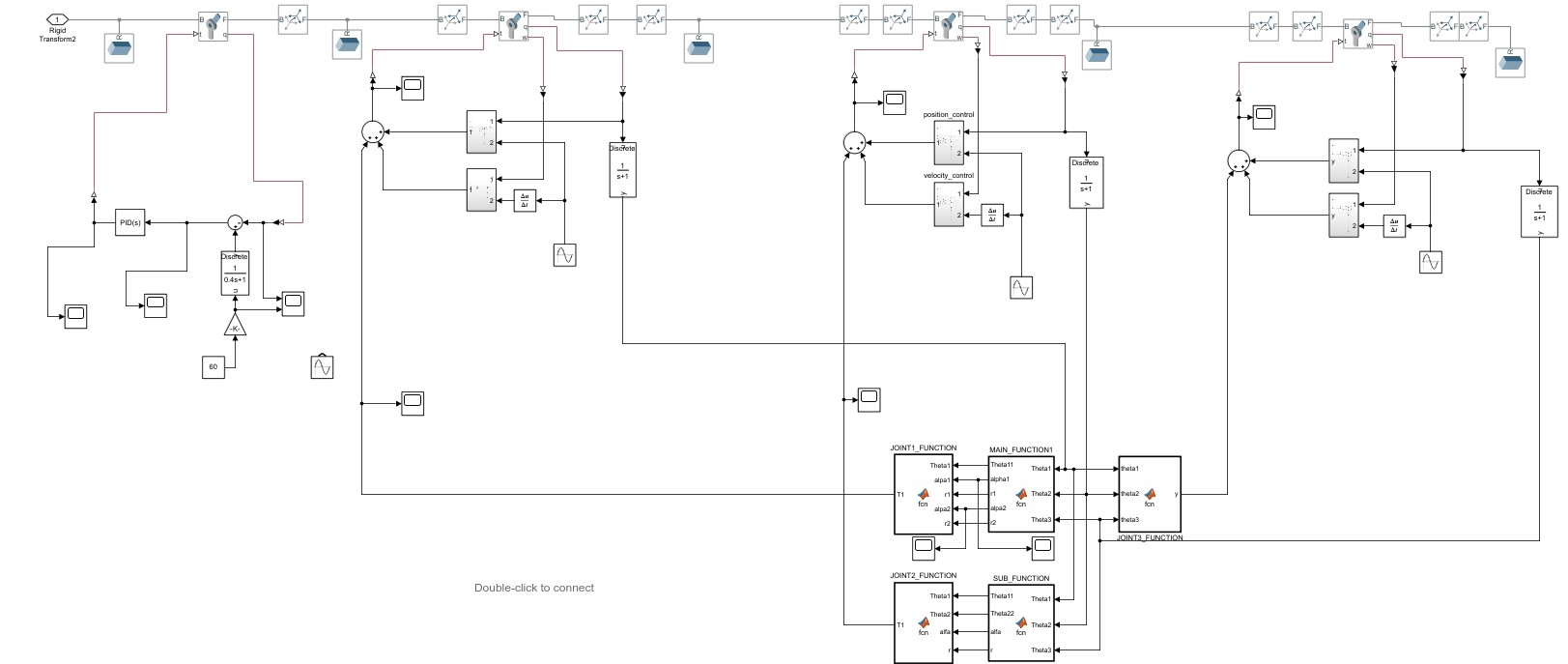


So , these remaining functions are responsible for calculating the torques on each joint .

* CREATING THE SIMULATION ENVIRONMENT :

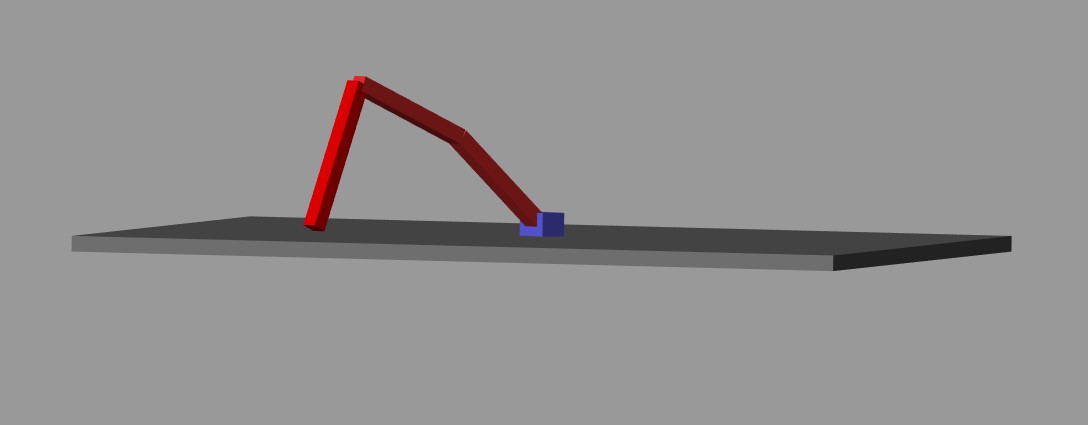
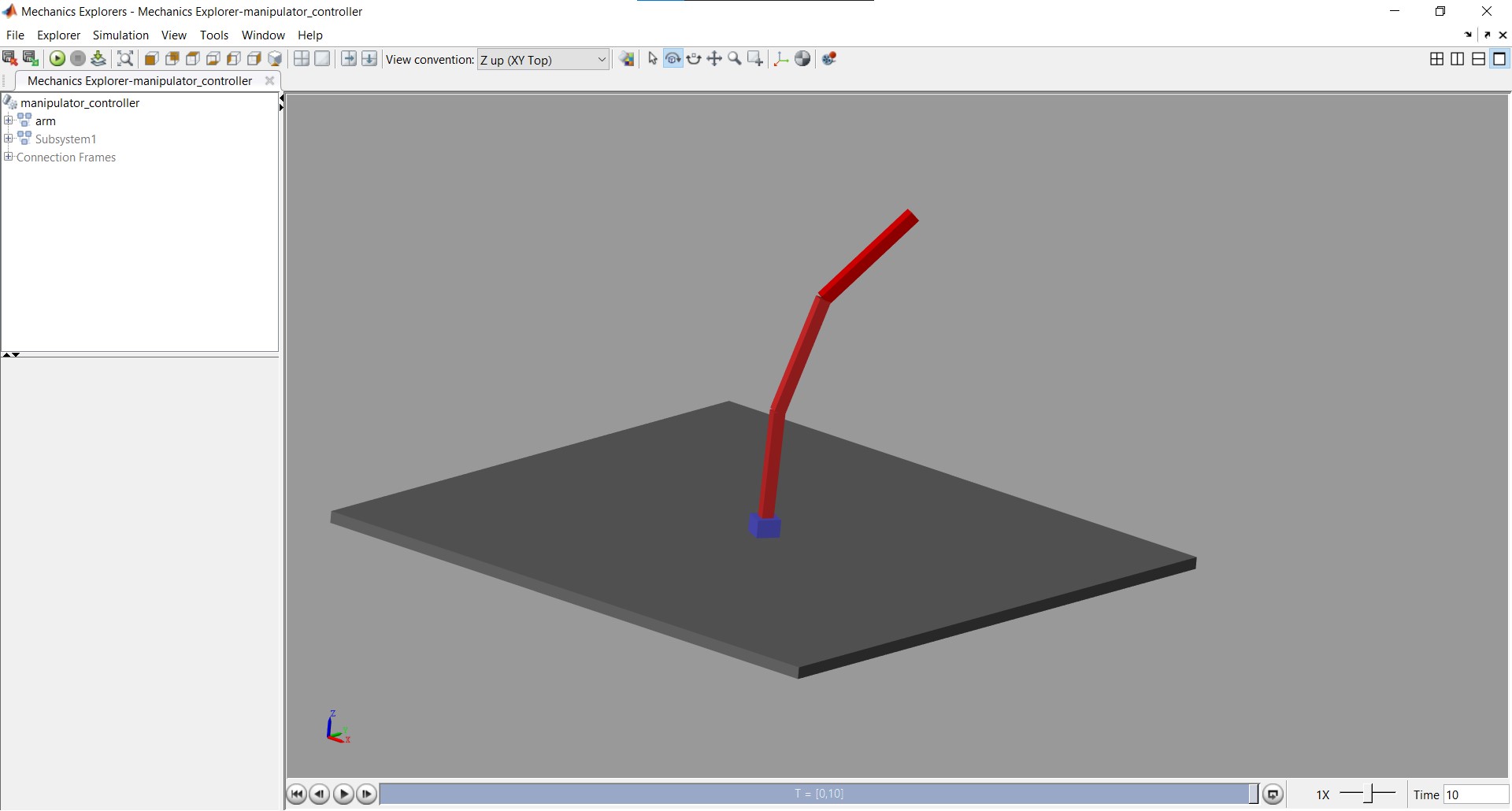
1. We created a manipulator , using simscape multibody to test our controller with each joint .

This is the overall view of the whole system .



We simulated using mechanics explorer in matlab ,

This is the final output of the simulation environment of the manipulator which we test . In the above file each joint had his own controller , to control the manipulator .



IMPLEMENTATION PHASE :

* In this phase we have to create an architecture for the manipulator controller , with appropriate microcontrollers .
* So , our architecture consists of two stm's, one responsible for controlling the end effector and another responsible for controlling below 3 joints .
* And one arduino nano as main controller for both stm’s for transferring data from the main communication node .

Stm controller code for end effector :

// ---------------------- - FINDING GRIPPER MOTOR TORQUE -------------------------------------

GRIPPER\_TORQUE = GRIPPER\_FEEDBACK\_I\*Kt; // MOTOR TORQUE

//---------------------------------------------------------------------------------------------------------------

// %%%%%%%%% GRIPPER TORQUE CONTROLLER %%%%%%%%%%%

GRIPPER\_ERROR = GRIPPER\_SET - GRIPPER\_TORQUE; // FINDING ERROR

GRIPPER\_PID = PID(GRIPPER\_ERROR, GRIPPER\_PREV\_ERROR, &GRIPPER\_I,

GRIPPER\_Kp, GRIPPER\_Ki, GRIPPER\_Kd, GRIPPER\_SAT\_PID, GRIPPER\_PID, G\_T); //

CALCULATING PID

GRIPPER\_PREV\_ERROR = GRIPPER\_ERROR;

GRIPPER\_SAT\_PID = SATURATE(GRIPPER\_PID,-SAT\_PID\_VALUE,SAT\_PID\_VALUE);

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

G\_T = micros();

This controller code is responsible for maintaining a constant gripping force on the gripper when activated .

It will get current data from the current sensors for the motor current to determine the torque .

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* YAW\_ANGULAR\_SPEED\_PID \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

YAW\_W\_ERROR = YAW\_SET\_W - YAW\_FEEDBACK\_W;

YAW\_PID\_W = PID( YAW\_W\_ERROR, YAW\_W\_PREV\_ERROR, &YAW\_W\_I, YAW\_W\_Kp,

YAW\_W\_Ki, YAW\_W\_Kd,YAW\_SAT\_PID\_W,YAW\_PID\_W,TIMER\_YAW);

YAW\_W\_PREV\_ERROR = YAW\_W\_ERROR;

YAW\_SAT\_PID\_W = SATURATE(YAW\_PID\_W,-SAT\_PID\_VALUE,SAT\_PID\_VALUE);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* YAW\_POSITION\_PID \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

YAW\_T\_ERROR = (YAW\_SET\_T - YAW\_FEEDBACK\_T)\*0.1 - YAW\_FEEDBACK\_I;

YAW\_PID\_T = PID( YAW\_T\_ERROR,YAW\_T\_PREV\_ERROR, &YAW\_T\_I, YAW\_T\_Kp,

YAW\_T\_Ki, YAW\_T\_Kd,YAW\_SAT\_PID\_T,YAW\_PID\_T,TIMER\_YAW);

YAW\_T\_PREV\_ERROR = YAW\_T\_ERROR;

YAW\_SAT\_PID\_T = SATURATE(YAW\_PID\_T,-SAT\_PID\_VALUE,SAT\_PID\_VALUE);

TIMER\_YAW = micros();

The above controller , had a position controller and a speed controller which is responsible for which appropriately control the joint angle . The position controller uses current based control .

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* JOINT\_5\_POSITION\_PID \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JOINT\_5\_T\_ERROR = (JOINT\_5\_SET\_T - JOINT\_5\_FEEDBACK\_T)\*0.1 -

JOINT\_5\_FEEDBACK\_I;

JOINT\_5\_PID = PID( JOINT\_5\_T\_ERROR, JOINT\_5\_T\_PREV\_ERROR, &JOINT\_5\_T\_I,

JOINT\_5\_T\_Kp, JOINT\_5\_T\_Ki, JOINT\_5\_T\_Kd,JOINT\_5\_SAT\_PID,JOINT\_5\_PID,TIMER\_5);

JOINT\_5\_T\_PREV\_ERROR = JOINT\_5\_T\_ERROR;

JOINT\_5\_SAT\_PID = SATURATE(JOINT\_5\_PID,-SAT\_PID\_VALUE,SAT\_PID\_VALUE);

TIMER\_5 = micros();

This joint 5 and 4 controller had only a current based position controller , because it will gate the commands from an inverse kinematic controller inside the main controller .

* When the rover is in autonomous mode , we can't give instructions to the manipulator from the base station , so the prosser has to give the data then in this mode each joint controller will implement only the current based position controller .
* When we are controlling the manipulator from the base station, then we only get the speed of each joint rather than position , at this time we will implement the speed controller along with the position controller for three joints .
* But for joints 5, 6 they always implement only with current based position controllers .

INVERSE KINEMATIC CODE :

* This inverse kinematic code is implemented in the main controller

////////////////////// FORWARD KINEMATICS /////////////////////////////

1. = L1\*sin(THETA4) + L2\*sin(THETA5) + double(CAN\_J\_4)/F;
2. = L1\*cos(THETA4) + L2\*sin(THETA5) + double(CAN\_J\_5)/F;

///////////////////// INVERSE KINEMATICS ///////////////////////////////

T2 = acos((X^2 + Y^2 - L1^2 - L2^2)/(2\*L1\*L2));

J4\_SET = T2;

K1 = L1 + L2\*cos(T2);

K2 = L2\*sin(T2);

T1 = atan2(X/Y) - atan2(K1/K2);

J5\_SET = T1;

We are only determining the two joint angles from the inverse kinematics.

In forward kinematics , we are updating the X and Y positions from user commands . Then we can apply inverse kinematics to determine the angles .

So the user has control over X , Y , YAW , END EFFECTOR ORIENTATION , JOINT\_3 AND GRIPPER .

So , the user can move the manipulator in X and Y direction and he has to control the bottom joint to rotate the manipulator .

User has another three controls to control the end effector orientation and the gripper actions .