# **CODE FOR TRANSMITTER-RECEIVER**

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h> //https://github.com/tmrh20/RF24/

#include <Wire.h>

#define PSB\_PAD\_DOWN PB1

#define PSB\_PAD\_LEFT PC15

#define PSB\_PAD\_UP PC14

#define PSB\_PAD\_RIGHT PB10

#define PSB\_TRIANGLE PB9

#define PSB\_SQUARE PB7

#define PSB\_CIRCLE PB5

#define PSB\_CROSS PB3

#define PSB\_START PB8

#define PSB\_SELECT PB6

#define PSB\_L1 PA9

#define PSB\_L2 PA10

RF24 radio(PB0,PA4); // nRF24L01 (CE, CSN)

const byte address[6] = "00001"; // Address

struct Data\_Package {

byte PSB\_PAD\_DOWN;

byte PSB\_PAD\_LEFT;

byte PSB\_PAD\_UP;

byte PSB\_PAD\_RIGHT;

byte PSB\_TRIANGLE;

byte PSB\_SQUARE;

byte PSB\_CIRCLE;

byte PSB\_CROSS;

byte PSB\_START;

byte PSB\_SELECT;

byte PSB\_L1;

byte PSB\_L2;

//JOY STICKS

int PSS\_RX;

int PSS\_RY;

int PSS\_LX;

int PSS\_LY;

};

Data\_Package data; //Create a variable with the above structure

void setup(){

Serial.begin(9600);

// radio communication

radio.begin();

radio.openWritingPipe(address);

radio.setAutoAck(false);

radio.setDataRate(RF24\_250KBPS);

radio.setPALevel(RF24\_PA\_LOW);

radio.stopListening();

pinMode(PSB\_PAD\_DOWN,INPUT\_PULLUP);

pinMode(PSB\_PAD\_UP,INPUT\_PULLUP);

pinMode(PSB\_PAD\_LEFT,INPUT\_PULLUP);

pinMode(PSB\_PAD\_RIGHT,INPUT\_PULLUP);

pinMode(PSB\_TRIANGLE,INPUT\_PULLUP);

pinMode(PSB\_SQUARE,INPUT\_PULLUP);

pinMode(PSB\_CIRCLE,INPUT\_PULLUP);

pinMode(PSB\_CROSS,INPUT\_PULLUP);

pinMode(PSB\_START,INPUT\_PULLUP);

pinMode(PSB\_SELECT,INPUT\_PULLUP);

pinMode(PSB\_L1,INPUT\_PULLUP);

pinMode(PSB\_L2,INPUT\_PULLUP);

/\*

pinMode(PSS\_RX,INPUT);

pinMode(PSS\_RY,INPUT);

pinMode(PSS\_LY,INPUT);

pinMode(PSS\_LX,INPUT);

\*/

// initial default values

data.PSB\_PAD\_DOWN = 1;

data.PSB\_PAD\_UP = 1;

data.PSB\_PAD\_RIGHT = 1;

data.PSB\_PAD\_LEFT = 1;

data.PSB\_CIRCLE = 1;

data.PSB\_TRIANGLE = 1;

data.PSB\_CROSS = 1;

data.PSB\_SQUARE = 1;

data.PSB\_SELECT = 1;

data.PSB\_START = 1;

data.PSB\_L1 = 1;

data.PSB\_L2 = 1;

data.PSS\_RX =0 ;

data.PSS\_RY = 0;

data.PSS\_LX = 0;

data.PSS\_LY =0;

}

void loop(){

data.PSS\_LY = angle(map(analogRead(PA2),0,4095,0,255));

data.PSS\_LX = angle(map(analogRead(PA3),0,4095,0,255));

data.PSS\_RX = angle(map(analogRead(PA1),0,4095,0,255));

data.PSS\_RY = angle(map(analogRead(PA0),0,4095,0,255));

// Read all digital inputs

data.PSB\_PAD\_DOWN = digitalRead(PSB\_PAD\_DOWN);

data.PSB\_PAD\_UP = digitalRead(PSB\_PAD\_UP);

data.PSB\_PAD\_RIGHT = digitalRead(PSB\_PAD\_RIGHT);

data.PSB\_PAD\_LEFT = digitalRead(PSB\_PAD\_LEFT);

data.PSB\_CIRCLE = digitalRead(PSB\_CIRCLE);

data.PSB\_TRIANGLE = digitalRead(PSB\_TRIANGLE);

data.PSB\_CROSS = digitalRead(PSB\_CROSS);

data.PSB\_SQUARE = digitalRead(PSB\_SQUARE);

data.PSB\_SELECT = digitalRead(PSB\_SELECT);

data.PSB\_START = digitalRead(PSB\_START);

data.PSB\_L1 = digitalRead(PSB\_L1);

data.PSB\_L2 = digitalRead(PSB\_L2);

Serial.print(data.PSB\_PAD\_DOWN );

Serial.print(" ");

Serial.print(data.PSB\_PAD\_UP );

Serial.print(" ");

Serial.print(data.PSB\_PAD\_RIGHT );

Serial.print(" ");

Serial.print(data.PSB\_PAD\_LEFT );

Serial.print(" ");

Serial.print(data.PSB\_TRIANGLE);

Serial.print(" ");

Serial.print(data.PSB\_CIRCLE);

Serial.print(" ");

Serial.print(data.PSB\_CROSS );

Serial.print(" > ");

Serial.print(data.PSB\_SQUARE );

Serial.print(" ");

Serial.print(data.PSB\_START );

Serial.print(" ");

Serial.print(data.PSB\_SELECT );

Serial.print(" ");

Serial.print(data.PSB\_L1 );

Serial.print(" ");

Serial.print(data.PSB\_L2 );

Serial.print(" ");

Serial.print(data.PSS\_RX);

Serial.print(" ");

Serial.print(data.PSS\_RY);

Serial.print(" ");

Serial.print(data.PSS\_LX);

Serial.print(" ");

Serial.print(data.PSS\_LY);

Serial.print(" ");

Serial.println();

radio.write(&data, sizeof(Data\_Package)); // Send the whole data from the structure to the receiver

}

int angle(int x){

if(x>0 && x<20)

{

x=0;

}

if(x>20 && x<40)

{

x=20;

}

if(x>40 && x<60)

{

x=40;

}

if(x>60 && x<80)

{

x=60;

}

if(x>80 && x<100)

{

x=80;

}

if(x>100 && x<120)

{

x=100;

}

if(x>120 && x<140)

{

x=120;

}

if(x>140 && x<160)

{

x=140;

}

if(x>160 && x<180)

{

x=160;

}

if(x>180 && x<200)

{

x=180;

}

if(x>200 && x<220)

{

x=200;

}

if(x>220 && x<240)

{

x=220;

}

else

{

x=x;

}

return x;

}

# **CODE FOR HEXAPOD ROBOT:**

#include <Servo.h>

#include <math.h>

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

#include <Adafruit\_PWMServoDriver.h>

#define COXA1\_SERVO 0 //servo port definitions

#define FEMUR1\_SERVO 1

#define TIBIA1\_SERVO 2

#define COXA2\_SERVO 3

#define FEMUR2\_SERVO 4

#define TIBIA2\_SERVO 5

#define COXA3\_SERVO 6

#define FEMUR3\_SERVO 7

#define TIBIA3\_SERVO 8

#define COXA4\_SERVO 9

#define FEMUR4\_SERVO 10

#define TIBIA4\_SERVO 11

#define COXA5\_SERVO PA0

#define FEMUR5\_SERVO PA1

#define TIBIA5\_SERVO PA2

#define COXA6\_SERVO 12

#define FEMUR6\_SERVO 13

#define TIBIA6\_SERVO 14

const int COXA\_LENGTH = 51; //leg part lengths

const int FEMUR\_LENGTH = 65;

const int TIBIA\_LENGTH = 121;

const int TRAVEL = 30; //translate and rotate travel limit constant

const long A12DEG = 209440; //12 degrees in radians x 1,000,000

const long A30DEG = 523599; //30 degrees in radians x 1,000,000

const int FRAME\_TIME\_MS = 20; //frame time (20msec = 50Hz)

const float HOME\_X[6] = { 82.0, 0.0, -82.0, -82.0, 0.0, 82.0}; //coxa-to-toe home positions

const float HOME\_Y[6] = { 82.0, 116.0, 82.0, -82.0, -116.0, -82.0};

const float HOME\_Z[6] = { -80.0, -80.0, -80.0, -80.0, -80.0, -80.0};

const float BODY\_X[6] = { 110.4, 0.0, -110.4, -110.4, 0.0, 110.4}; //body center-to-coxa servo distances

const float BODY\_Y[6] = { 58.4, 90.8, 58.4, -58.4, -90.8, -58.4};

const float BODY\_Z[6] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

const int COXA\_CAL[6] = {2, -1, -1, -3, -2, -3}; //servo calibration constants

const int FEMUR\_CAL[6] = {4, -2, 0, -1, 0, 0};

const int TIBIA\_CAL[6] = {0, -3, -3, -2, -3, -1};

RF24 radio(PB0, PA4); // nRF24L01 (CE, CSN)

const byte address[6] = "00001";

unsigned long lastReceiveTime = 0;

unsigned long currentTime = 0;

int gamepad\_error; //gamepad variables

byte gamepad\_type;

byte gamepad\_vibrate;

//unsigned long currentTime; //frame timer variables

unsigned long previousTime;

int temp; //mode and control variables

int mode;

int gait;

int gait\_speed;

int gait\_LED\_color;

int reset\_position;

int capture\_offsets;

int batt\_LEDs; //battery monitor variables

int batt\_voltage;

int batt\_voltage\_index;

int batt\_voltage\_array[50];

long batt\_voltage\_sum;

float L0, L3; //inverse kinematics variables

float gamma\_femur;

float phi\_tibia, phi\_femur;

float theta\_tibia, theta\_femur, theta\_coxa;

int leg1\_IK\_control, leg6\_IK\_control; //leg lift mode variables

float leg1\_coxa, leg1\_femur, leg1\_tibia;

float leg6\_coxa, leg6\_femur, leg6\_tibia;

int leg\_num; //positioning and walking variables

int z\_height\_LED\_color;

int totalX, totalY, totalZ;

int tick, duration, numTicks;

int z\_height\_left, z\_height\_right;

int commandedX, commandedY, commandedR;

int translateX, translateY, translateZ;

float step\_height\_multiplier;

float strideX, strideY, strideR;

float sinRotX, sinRotY, sinRotZ;

float cosRotX, cosRotY, cosRotZ;

float rotOffsetX, rotOffsetY, rotOffsetZ;

float amplitudeX, amplitudeY, amplitudeZ;

float offset\_X[6], offset\_Y[6], offset\_Z[6];

float current\_X[6], current\_Y[6], current\_Z[6];

int tripod\_case[6] = {1,2,1,2,1,2}; //for tripod gait walking

int ripple\_case[6] = {2,6,4,1,3,5}; //for ripple gait

int wave\_case[6] = {1,2,3,4,5,6}; //for wave gait

int tetrapod\_case[6] = {1,3,2,1,2,3}; //for tetrapod gait

Servo coxa6\_servo;

Servo femur6\_servo;

Servo tibia6\_servo;

// Max size of this struct is 32 bytes - NRF24L01 buffer limit

struct Data\_Package {

//BUTTONS

byte PSB\_PAD\_DOWN;

byte PSB\_PAD\_LEFT;

byte PSB\_PAD\_UP;

byte PSB\_PAD\_RIGHT;

byte PSB\_TRIANGLE;

byte PSB\_SQUARE;

byte PSB\_CIRCLE;

byte PSB\_CROSS;

byte PSB\_START;

byte PSB\_SELECT;

byte PSB\_L1;

byte PSB\_L2;

//JOY STICKS

int PSS\_RY;

int PSS\_RX;

int PSS\_LX;

int PSS\_LY;

};

Data\_Package data; //Create a variable with the above structure

Adafruit\_PWMServoDriver pca= Adafruit\_PWMServoDriver(0x40);

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

// Initialization Routine

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

void setup()

{

//start serial

Serial.begin(11500);

pca.begin();

pca.setPWMFreq(60);

//attach servos

coxa6\_servo.attach(COXA6\_SERVO,610,2400);

femur6\_servo.attach(FEMUR6\_SERVO,610,2400);

tibia6\_servo.attach(TIBIA6\_SERVO,610,2400);

//NRF24 CONNECTION

radio.begin();

radio.openReadingPipe(0, address);

radio.setAutoAck(false);

radio.setDataRate(RF24\_250KBPS);

radio.setPALevel(RF24\_PA\_LOW);

radio.startListening(); // Set the module as receiver

resetData();

//clear offsets

for(leg\_num=0; leg\_num<6; leg\_num++)

{

offset\_X[leg\_num] = 0.0;

offset\_Y[leg\_num] = 0.0;

offset\_Z[leg\_num] = 0.0;

}

capture\_offsets = false;

step\_height\_multiplier = 1.0;

//initialize mode and gait variables

mode = 0;

gait = 0;

gait\_speed = 0;

reset\_position = true;

leg1\_IK\_control = true;

leg6\_IK\_control = true;

}

void loop()

{

if (radio.available())

{

radio.read(&data, sizeof(Data\_Package)); // Read the whole data and store it into the 'data' structure

lastReceiveTime = millis(); // At this moment we have received the data

}

if ( currentTime - lastReceiveTime > 1000 )

{ // If current time is more then 1 second since we have recived the last data, that means we have lost connection

resetData(); // If connection is lost, reset the data. It prevents unwanted behavior, for example if a drone has a throttle up and we lose connection, it can keep flying unless we reset the values

}

//set up frame time

currentTime = millis();

if((currentTime - previousTime) > FRAME\_TIME\_MS)

{

previousTime = currentTime;

process\_gamepad();

Serial.println();

Serial.print("mode");

Serial.print(mode);

Serial.print(" ");

Serial.print("gait");

Serial.print(gait);

Serial.print(" ");

Serial.print("RX");

Serial.print(data.PSS\_RX);

Serial.print(" ");

Serial.print("RY");

Serial.print(data.PSS\_RY);

Serial.print(" ");

Serial.print("LX");

Serial.print(data.PSS\_LX);

Serial.print(" ");

Serial.print("LY");

Serial.print(data.PSS\_LY);

//reset legs to home position when commanded

if(reset\_position == true)

{

for(leg\_num=0; leg\_num<6; leg\_num++)

{

current\_X[leg\_num] = HOME\_X[leg\_num];

current\_Y[leg\_num] = HOME\_Y[leg\_num];

current\_Z[leg\_num] = HOME\_Z[leg\_num];

}

reset\_position = false;

}

//position legs using IK calculations - unless set all to 90 degrees mode

if(mode < 99)

{

for(leg\_num=0; leg\_num<6; leg\_num++)

leg\_IK(leg\_num,current\_X[leg\_num]+offset\_X[leg\_num],current\_Y[leg\_num]+offset\_Y[leg\_num],current\_Z[leg\_num]+offset\_Z[leg\_num]);

}

//reset leg lift first pass flags if needed

if(mode != 4)

{

leg1\_IK\_control = true;

leg6\_IK\_control = true;

}

//battery\_monitor(); //battery monitor and output to LEDs

print\_debug(); //print debug data

//process modes (mode 0 is default 'home idle' do-nothing mode)

if(mode == 1) //walking mode

{

if(gait == 0) tripod\_gait(); //walk using gait 0

if(gait == 1) wave\_gait(); //walk using gait 1

if(gait == 2) ripple\_gait(); //walk using gait 2

if(gait == 3) tetrapod\_gait(); //walk using gait 3

}

if(mode == 2) translate\_control(); //joystick control x-y-z mode

if(mode == 3) rotate\_control(); //joystick control y-p-r mode

// if(mode == 4) one\_leg\_lift(); //one leg lift mode

if(mode == 99) set\_all\_90(); //set all servos to 90 degrees mode

}

}

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

// Process gamepad controller inputs

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

void process\_gamepad()

{

if(data.PSB\_PAD\_DOWN== 0) //stop & select gait 0

{

// Serial.print("down pressed");

mode = 0;

gait = 0;

reset\_position = true;

}

if(data.PSB\_PAD\_LEFT == 0) //stop & select gait 1

{

// Serial.print("left pressed");

mode = 0;

gait = 1;

reset\_position = true;

}

if(data.PSB\_PAD\_UP == 0) //stop & select gait 2

{

//Serial.print("up pressed");

mode = 0;

gait = 2;

reset\_position = true;

}

if(data.PSB\_PAD\_RIGHT == 0) //stop & select gait 3

{

//Serial.print("right pressed");

mode = 0;

gait = 3;

reset\_position = true;

}

if(data.PSB\_TRIANGLE == 0) //select walk mode

{

//Serial.print("triangle pressed");

mode = 1;

reset\_position = true;

}

if(data.PSB\_SQUARE == 0) //control x-y-z with joysticks mode

{

//Serial.print("square pressed");

mode = 2;

reset\_position = true;

}

if(data.PSB\_CIRCLE == 0) //control y-p-r with joysticks mode

{

//Serial.print("circle pressed");

mode = 3;

reset\_position = true;

}

if(data.PSB\_CROSS == 0) //one leg lift mode

{

//Serial.print("cross pressed");

mode = 4;

reset\_position = true;

}

if(data.PSB\_START == 0) //change gait speed

{

//Serial.print("start pressed");

if(gait\_speed == 0)

gait\_speed = 1;

else

gait\_speed = 0;

}

/\* if(PSB\_START == 1) //display gait speed on LEDs if button held

{

if(gait\_speed == 0) LED\_Bar(1,8); //use green LEDs for fast

else LED\_Bar(0,8); //use red LEDs for slow

}

\*/

if(data.PSB\_SELECT == 0) //set all servos to 90 degrees for calibration

{

//Serial.print("select pressed");

mode = 99;

}

if(data.PSB\_L1 == 0)

{

//Serial.print("L1 pressed");

//capture offsets in translate, rotate, and translate/rotate modes

capture\_offsets = true;

}

if(data.PSB\_L2 == 0)

{

//Serial.print("L2 pressed");

for(leg\_num=0; leg\_num<6; leg\_num++) //clear offsets

{

offset\_X[leg\_num] = 0;

offset\_Y[leg\_num] = 0;

offset\_Z[leg\_num] = 0;

}

leg1\_IK\_control = true; //reset leg lift first pass flags

leg6\_IK\_control = true;

step\_height\_multiplier = 1.0; //reset step height multiplier

}

}

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

// Leg IK Routine

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

void leg\_IK(int leg\_number,float X,float Y,float Z)

{

//compute target femur-to-toe (L3) length

L0 = sqrt(sq(X) + sq(Y)) - COXA\_LENGTH;

L3 = sqrt(sq(L0) + sq(Z));

//process only if reach is within possible range (not too long or too short!)

if((L3 < (TIBIA\_LENGTH+FEMUR\_LENGTH)) && (L3 > (TIBIA\_LENGTH-FEMUR\_LENGTH)))

{

//compute tibia angle

phi\_tibia = acos((sq(FEMUR\_LENGTH) + sq(TIBIA\_LENGTH) - sq(L3))/(2\*FEMUR\_LENGTH\*TIBIA\_LENGTH));

theta\_tibia = phi\_tibia\*RAD\_TO\_DEG - 23.0 + TIBIA\_CAL[leg\_number];

theta\_tibia = constrain(theta\_tibia,0.0,180.0);

//compute femur angle

gamma\_femur = atan2(Z,L0);

phi\_femur = acos((sq(FEMUR\_LENGTH) + sq(L3) - sq(TIBIA\_LENGTH))/(2\*FEMUR\_LENGTH\*L3));

theta\_femur = (phi\_femur + gamma\_femur)\*RAD\_TO\_DEG + 14.0 + 90.0 + FEMUR\_CAL[leg\_number];

theta\_femur = constrain(theta\_femur,0.0,180.0);

//compute coxa angle

theta\_coxa = atan2(X,Y)\*RAD\_TO\_DEG + COXA\_CAL[leg\_number];

//output to the appropriate leg

switch(leg\_number)

{

case 0:

if(leg1\_IK\_control == true) //flag for IK or manual control of leg

{

theta\_coxa = theta\_coxa + 45.0; //compensate for leg mounting

theta\_coxa = constrain(theta\_coxa,0.0,180.0);

pca.writeMicroseconds(COXA1\_SERVO,angle((int(theta\_coxa))));

pca.writeMicroseconds(FEMUR1\_SERVO,angle((int(theta\_femur))));

pca.writeMicroseconds(TIBIA1\_SERVO,angle(int(theta\_tibia)));

}

break;

case 1:

theta\_coxa = theta\_coxa + 90.0; //compensate for leg mounting

theta\_coxa = constrain(theta\_coxa,0.0,180.0);

pca.writeMicroseconds(COXA2\_SERVO,angle((int(theta\_coxa))));

pca.writeMicroseconds(FEMUR2\_SERVO,angle((int(theta\_femur))));

pca.writeMicroseconds(TIBIA2\_SERVO,angle(int(theta\_tibia)));

break;

case 2:

theta\_coxa = theta\_coxa + 135.0; //compensate for leg mounting

theta\_coxa = constrain(theta\_coxa,0.0,180.0);

pca.writeMicroseconds(COXA3\_SERVO,angle((int(theta\_coxa))));

pca.writeMicroseconds(FEMUR3\_SERVO,angle((int(theta\_femur))));

pca.writeMicroseconds(TIBIA3\_SERVO,angle((int(theta\_tibia))));

break;

case 3:

if(theta\_coxa < 0) //compensate for leg mounting

theta\_coxa = theta\_coxa + 225.0; // (need to use different

else // positive and negative offsets

theta\_coxa = theta\_coxa - 135.0; // due to atan2 results above!)

theta\_coxa = constrain(theta\_coxa,0.0,180.0);

pca.writeMicroseconds(COXA4\_SERVO,angle((int(theta\_coxa))));

pca.writeMicroseconds(FEMUR4\_SERVO,angle((int(theta\_femur))));

pca.writeMicroseconds(TIBIA4\_SERVO,angle(int(theta\_tibia)));

break;

case 4:

if(theta\_coxa < 0) //compensate for leg mounting

theta\_coxa = theta\_coxa + 270.0; // (need to use different

else // positive and negative offsets

theta\_coxa = theta\_coxa - 90.0; // due to atan2 results above!)

theta\_coxa = constrain(theta\_coxa,0.0,180.0);

pca.writeMicroseconds(COXA5\_SERVO,angle((int(theta\_coxa))));

pca.writeMicroseconds(FEMUR5\_SERVO,angle((int(theta\_femur))));

pca.writeMicroseconds(TIBIA5\_SERVO,angle(int(theta\_tibia)));

break;

case 5:

if(leg6\_IK\_control == true) //flag for IK or manual control of leg

{

if(theta\_coxa < 0) //compensate for leg mounting

theta\_coxa = theta\_coxa + 315.0; // (need to use different

else // positive and negative offsets

theta\_coxa = theta\_coxa - 45.0; // due to atan2 results above!)

theta\_coxa = constrain(theta\_coxa,0.0,180.0);

coxa6\_servo.write(int(theta\_coxa));

femur6\_servo.write(int(theta\_femur));

tibia6\_servo.write(int(theta\_tibia));

}

break;

}

}

}

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

// Tripod Gait

// Group of 3 legs move forward while the other 3 legs provide support

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

void tripod\_gait()

{

//read commanded values from controller

commandedX = map(data.PSS\_RY,0,255,127,-127);

commandedY = map(data.PSS\_RX,0,255,-127,127);

commandedR = map(data.PSS\_LX,0,255,127,-127);

//if commands more than deadband then process

if((abs(commandedX) > 40) || (abs(commandedY) > 40) || (abs(commandedR) > 40) || (tick>0))

{

compute\_strides();

numTicks = round(duration / FRAME\_TIME\_MS / 2.0); //total ticks divided into the two cases

for(leg\_num=0; leg\_num<6; leg\_num++)

{

compute\_amplitudes();

switch(tripod\_case[leg\_num])

{

case 1: //move foot forward (raise and lower)

current\_X[leg\_num] = HOME\_X[leg\_num] - amplitudeX\*cos(M\_PI\*tick/numTicks);

current\_Y[leg\_num] = HOME\_Y[leg\_num] - amplitudeY\*cos(M\_PI\*tick/numTicks);

current\_Z[leg\_num] = HOME\_Z[leg\_num] + abs(amplitudeZ)\*sin(M\_PI\*tick/numTicks);

if(tick >= numTicks-1) tripod\_case[leg\_num] = 2;

break;

case 2: //move foot back (on the ground)

current\_X[leg\_num] = HOME\_X[leg\_num] + amplitudeX\*cos(M\_PI\*tick/numTicks);

current\_Y[leg\_num] = HOME\_Y[leg\_num] + amplitudeY\*cos(M\_PI\*tick/numTicks);

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) tripod\_case[leg\_num] = 1;

break;

}

}

//increment tick

if(tick < numTicks-1) tick++;

else tick = 0;

}

}

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

// Wave Gait

// Legs move forward one at a time while the other 5 legs provide support

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

void wave\_gait()

{

//read commanded values from controller

commandedX = map(data.PSS\_RY,0,255,127,-127);

commandedY = map(data.PSS\_RX,0,255,-127,127);

commandedR = map(data.PSS\_LX,0,255,127,-127);

//if commands more than deadband then process

if((abs(commandedX) > 40) || (abs(commandedY) > 40) || (abs(commandedR) > 40) || (tick>0))

{

compute\_strides();

numTicks = round(duration / FRAME\_TIME\_MS / 6.0); //total ticks divided into the six cases

for(leg\_num=0; leg\_num<6; leg\_num++)

{

compute\_amplitudes();

switch(wave\_case[leg\_num])

{

case 1: //move foot forward (raise and lower)

current\_X[leg\_num] = HOME\_X[leg\_num] - amplitudeX\*cos(M\_PI\*tick/numTicks);

current\_Y[leg\_num] = HOME\_Y[leg\_num] - amplitudeY\*cos(M\_PI\*tick/numTicks);

current\_Z[leg\_num] = HOME\_Z[leg\_num] + abs(amplitudeZ)\*sin(M\_PI\*tick/numTicks);

if(tick >= numTicks-1) wave\_case[leg\_num] = 6;

break;

case 2: //move foot back one-fifth (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.5;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.5;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) wave\_case[leg\_num] = 1;

break;

case 3: //move foot back one-fifth (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.5;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.5;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) wave\_case[leg\_num] = 2;

break;

case 4: //move foot back one-fifth (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.5;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.5;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1)

wave\_case[leg\_num] = 3;

break;

case 5: //move foot back one-fifth (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.5;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.5;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) wave\_case[leg\_num] = 4;

break;

case 6: //move foot back one-fifth (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.5;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.5;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) wave\_case[leg\_num] = 5;

break;

}

}

//increment tick

if(tick < numTicks-1) tick++;

else tick = 0;

}

}

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

// Ripple Gait

// Left legs move forward rear-to-front while right also do the same,

// but right side is offset so RR starts midway through the LM stroke

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

void ripple\_gait()

{

//read commanded values from controller

commandedX = map(data.PSS\_RY,0,255,127,-127);

commandedY = map(data.PSS\_RX,0,255,-127,127);

commandedR = map(data.PSS\_LX,0,255,127,-127);

//if commands more than deadband then process

if((abs(commandedX) > 40) || (abs(commandedY) > 40) || (abs(commandedR) > 40) || (tick>0))

{

compute\_strides();

numTicks = round(duration / FRAME\_TIME\_MS / 6.0); //total ticks divided into the six cases

for(leg\_num=0; leg\_num<6; leg\_num++)

{

compute\_amplitudes();

switch(ripple\_case[leg\_num])

{

case 1: //move foot forward (raise)

current\_X[leg\_num] = HOME\_X[leg\_num] - amplitudeX\*cos(M\_PI\*tick/(numTicks\*2));

current\_Y[leg\_num] = HOME\_Y[leg\_num] - amplitudeY\*cos(M\_PI\*tick/(numTicks\*2));

current\_Z[leg\_num] = HOME\_Z[leg\_num] + abs(amplitudeZ)\*sin(M\_PI\*tick/(numTicks\*2));

if(tick >= numTicks-1) ripple\_case[leg\_num] = 2;

break;

case 2: //move foot forward (lower)

current\_X[leg\_num] = HOME\_X[leg\_num] - amplitudeX\*cos(M\_PI\*(numTicks+tick)/(numTicks\*2));

current\_Y[leg\_num] = HOME\_Y[leg\_num] - amplitudeY\*cos(M\_PI\*(numTicks+tick)/(numTicks\*2));

current\_Z[leg\_num] = HOME\_Z[leg\_num] + abs(amplitudeZ)\*sin(M\_PI\*(numTicks+tick)/(numTicks\*2));

if(tick >= numTicks-1) ripple\_case[leg\_num] = 3;

break;

case 3: //move foot back one-quarter (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.0;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.0;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) ripple\_case[leg\_num] = 4;

break;

case 4: //move foot back one-quarter (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.0;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.0;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) ripple\_case[leg\_num] = 5;

break;

case 5: //move foot back one-quarter (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.0;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.0;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) ripple\_case[leg\_num] = 6;

break;

case 6: //move foot back one-quarter (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks/2.0;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks/2.0;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) ripple\_case[leg\_num] = 1;

break;

}

}

//increment tick

if(tick < numTicks-1) tick++;

else tick = 0;

}

}

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

// Tetrapod Gait

// Right front and left rear legs move forward together, then right

// rear and left middle, and finally right middle and left front.

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

void tetrapod\_gait()

{

//read commanded values from controller

commandedX = map(data.PSS\_RY,0,255,127,-127);

commandedY = map(data.PSS\_RX,0,255,-127,127);

commandedR = map(data.PSS\_LX,0,255,127,-127);

//if commands more than deadband then process

if((abs(commandedX) > 40) || (abs(commandedY) > 40) || (abs(commandedR) > 40) || (tick>0))

{

compute\_strides();

numTicks = round(duration / FRAME\_TIME\_MS / 3.0); //total ticks divided into the three cases

for(leg\_num=0; leg\_num<6; leg\_num++)

{

compute\_amplitudes();

switch(tetrapod\_case[leg\_num])

{

case 1: //move foot forward (raise and lower)

current\_X[leg\_num] = HOME\_X[leg\_num] - amplitudeX\*cos(M\_PI\*tick/numTicks);

current\_Y[leg\_num] = HOME\_Y[leg\_num] - amplitudeY\*cos(M\_PI\*tick/numTicks);

current\_Z[leg\_num] = HOME\_Z[leg\_num] + abs(amplitudeZ)\*sin(M\_PI\*tick/numTicks);

if(tick >= numTicks-1) tetrapod\_case[leg\_num] = 2;

break;

case 2: //move foot back one-half (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) tetrapod\_case[leg\_num] = 3;

break;

case 3: //move foot back one-half (on the ground)

current\_X[leg\_num] = current\_X[leg\_num] - amplitudeX/numTicks;

current\_Y[leg\_num] = current\_Y[leg\_num] - amplitudeY/numTicks;

current\_Z[leg\_num] = HOME\_Z[leg\_num];

if(tick >= numTicks-1) tetrapod\_case[leg\_num] = 1;

break;

}

}

//increment tick

if(tick < numTicks-1) tick++;

else tick = 0;

}

}

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

// Compute walking stride lengths

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

void compute\_strides()

{

//compute stride lengths

strideX = 90\*commandedX/127;

strideY = 90\*commandedY/127;

strideR = 35\*commandedR/127;

//compute rotation trig

sinRotZ = sin(radians(strideR));

cosRotZ = cos(radians(strideR));

//set duration for normal and slow speed modes

if(gait\_speed == 0) duration = 1080;

else duration = 3240;

}

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

// Compute walking amplitudes

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

void compute\_amplitudes()

{

//compute total distance from center of body to toe

totalX = HOME\_X[leg\_num] + BODY\_X[leg\_num];

totalY = HOME\_Y[leg\_num] + BODY\_Y[leg\_num];

//compute rotational offset

rotOffsetX = totalY\*sinRotZ + totalX\*cosRotZ - totalX;

rotOffsetY = totalY\*cosRotZ - totalX\*sinRotZ - totalY;

//compute X and Y amplitude and constrain to prevent legs from crashing into each other

amplitudeX = ((strideX + rotOffsetX)/2.0);

amplitudeY = ((strideY + rotOffsetY)/2.0);

amplitudeX = constrain(amplitudeX,-50,50);

amplitudeY = constrain(amplitudeY,-50,50);

//compute Z amplitude

if(abs(strideX + rotOffsetX) > abs(strideY + rotOffsetY))

amplitudeZ = step\_height\_multiplier \* (strideX + rotOffsetX) /4.0;

else

amplitudeZ = step\_height\_multiplier \* (strideY + rotOffsetY) / 4.0;

}

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

// Body translate with controller (xyz axes)

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

void translate\_control()

{

//compute X direction move

translateX = map(data.PSS\_RY,0,255,-2\*TRAVEL,2\*TRAVEL);

for(leg\_num=0; leg\_num<6; leg\_num++)

current\_X[leg\_num] = HOME\_X[leg\_num] + translateX;

//compute Y direction move

translateY = map(data.PSS\_RX,0,255,2\*TRAVEL,-2\*TRAVEL);

for(leg\_num=0; leg\_num<6; leg\_num++)

current\_Y[leg\_num] = HOME\_Y[leg\_num] + translateY;

//compute Z direction move

translateZ = data.PSS\_LY;

if(translateZ > 127)

translateZ = map(translateZ,128,255,0,TRAVEL);

else

translateZ = map(translateZ,0,127,-3\*TRAVEL,0);

for(leg\_num=0; leg\_num<6; leg\_num++)

current\_Z[leg\_num] = HOME\_Z[leg\_num] + translateZ;

//lock in offsets if commanded

if(capture\_offsets == true)

{

for(leg\_num=0; leg\_num<6; leg\_num++)

{

offset\_X[leg\_num] = offset\_X[leg\_num] + translateX;

offset\_Y[leg\_num] = offset\_Y[leg\_num] + translateY;

offset\_Z[leg\_num] = offset\_Z[leg\_num] + translateZ;

current\_X[leg\_num] = HOME\_X[leg\_num];

current\_Y[leg\_num] = HOME\_Y[leg\_num];

current\_Z[leg\_num] = HOME\_Z[leg\_num];

}

}

//if offsets were commanded, exit current mode

if(capture\_offsets == true)

{

capture\_offsets = false;

mode = 0;

}

}

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

// Body rotate with controller (xyz axes)

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

void rotate\_control()

{

//compute rotation sin/cos values using controller inputs

sinRotX = sin((map(data.PSS\_RX,0,255,A12DEG,-A12DEG))/1000000.0);

cosRotX = cos((map(data.PSS\_RX,0,255,A12DEG,-A12DEG))/1000000.0);

sinRotY = sin((map(data.PSS\_RY,0,255,A12DEG,-A12DEG))/1000000.0);

cosRotY = cos((map(data.PSS\_RY,0,255,A12DEG,-A12DEG))/1000000.0);

sinRotZ = sin((map(data.PSS\_LX,0,255,-A30DEG,A30DEG))/1000000.0);

cosRotZ = cos((map(data.PSS\_LX,0,255,-A30DEG,A30DEG))/1000000.0);

//compute Z direction move

translateZ = data.PSS\_LY;

if(translateZ > 127)

translateZ = map(translateZ,128,255,0,TRAVEL);

else

translateZ = map(translateZ,0,127,-3\*TRAVEL,0);

for(int leg\_num=0; leg\_num<6; leg\_num++)

{

//compute total distance from center of body to toe

totalX = HOME\_X[leg\_num] + BODY\_X[leg\_num];

totalY = HOME\_Y[leg\_num] + BODY\_Y[leg\_num];

totalZ = HOME\_Z[leg\_num] + BODY\_Z[leg\_num];

//perform 3 axis rotations

rotOffsetX = totalX\*cosRotY\*cosRotZ + totalY\*sinRotX\*sinRotY\*cosRotZ + totalY\*cosRotX\*sinRotZ - totalZ\*cosRotX\*sinRotY\*cosRotZ + totalZ\*sinRotX\*sinRotZ - totalX;

rotOffsetY = -totalX\*cosRotY\*sinRotZ - totalY\*sinRotX\*sinRotY\*sinRotZ + totalY\*cosRotX\*cosRotZ + totalZ\*cosRotX\*sinRotY\*sinRotZ + totalZ\*sinRotX\*cosRotZ - totalY;

rotOffsetZ = totalX\*sinRotY - totalY\*sinRotX\*cosRotY + totalZ\*cosRotX\*cosRotY - totalZ;

// Calculate foot positions to achieve desired rotation

current\_X[leg\_num] = HOME\_X[leg\_num] + rotOffsetX;

current\_Y[leg\_num] = HOME\_Y[leg\_num] + rotOffsetY;

current\_Z[leg\_num] = HOME\_Z[leg\_num] + rotOffsetZ + translateZ;

//lock in offsets if commanded

if(capture\_offsets == true)

{

offset\_X[leg\_num] = offset\_X[leg\_num] + rotOffsetX;

offset\_Y[leg\_num] = offset\_Y[leg\_num] + rotOffsetY;

offset\_Z[leg\_num] = offset\_Z[leg\_num] + rotOffsetZ + translateZ;

current\_X[leg\_num] = HOME\_X[leg\_num];

current\_Y[leg\_num] = HOME\_Y[leg\_num];

current\_Z[leg\_num] = HOME\_Z[leg\_num];

}

}

//if offsets were commanded, exit current mode

if(capture\_offsets == true)

{

capture\_offsets = false;

mode = 0;

}

}

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

// One leg lift mode

// also can set z step height using capture offsets

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

/\*

void one\_leg\_lift()

{

//read current leg servo 1 positions the first time

if(leg1\_IK\_control == true)

{

leg1\_coxa = coxa1\_servo.read();

leg1\_femur = femur1\_servo.read();

leg1\_tibia = tibia1\_servo.read();

leg1\_IK\_control = false;

}

//read current leg servo 6 positions the first time

if(leg6\_IK\_control == true)

{

leg6\_coxa = coxa6\_servo.read();

leg6\_femur = femur6\_servo.read();

leg6\_tibia = tibia6\_servo.read();

leg6\_IK\_control = false;

}

//process right joystick left/right axis

temp = data.PSS\_RX;

temp = map(temp,0,255,45,-45);

coxa1\_servo.write(constrain(int(leg1\_coxa+temp),45,135));

//process right joystick up/down axis

temp = data.PSS\_RY;

if(temp < 117) //if joystick moved up

{

temp = map(temp,116,0,0,24); //move leg 1

femur1\_servo.write(constrain(int(leg1\_femur+temp),0,170));

tibia1\_servo.write(constrain(int(leg1\_tibia+4\*temp),0,170));

}

else //if joystick moved down

{

z\_height\_right = constrain(temp,140,255); //set Z step height

z\_height\_right = map(z\_height\_right,140,255,1,8);

}

//process left joystick left/right axis

temp = data.PSS\_LX;

temp = map(temp,0,255,45,-45);

coxa6\_servo.write(constrain(int(leg6\_coxa+temp),45,135));

//process left joystick up/down axis

temp = data.PSS\_LY;

if(temp < 117) //if joystick moved up

{

temp = map(temp,116,0,0,24); //move leg 6

femur6\_servo.write(constrain(int(leg6\_femur+temp),0,170));

tibia6\_servo.write(constrain(int(leg6\_tibia+4\*temp),0,170));

}

else //if joystick moved down

{

z\_height\_left = constrain(temp,140,255); //set Z step height

z\_height\_left = map(z\_height\_left,140,255,1,8);

}

//process z height adjustment

if(z\_height\_left>z\_height\_right)

z\_height\_right = z\_height\_left; //use max left or right value

/\* if(batt\_LEDs > 3) z\_height\_LED\_color=0; //use red LEDs if battery strong

else z\_height\_LED\_color=1; //use green LEDs if battery weak

LED\_Bar(z\_height\_LED\_color,z\_height\_right); //display Z height

if(capture\_offsets == true) //lock in Z height if commanded

{

step\_height\_multiplier = 1.0 + ((z\_height\_right - 1.0) / 3.0);

capture\_offsets = false;

}

}

\*/

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

// Set all servos to 90 degrees

// Note: this is useful for calibration/alignment of the servos

// i.e: set COXA\_CAL[6], FEMUR\_CAL[6], and TIBIA\_CAL[6] values in

// constants section above so all angles appear as 90 degrees

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

void set\_all\_90()

{

pca.writeMicroseconds(COXA1\_SERVO,angle((90+COXA\_CAL[0])));

pca.writeMicroseconds(FEMUR1\_SERVO,angle((90+FEMUR\_CAL[0])));

pca.writeMicroseconds(TIBIA1\_SERVO,angle((90+TIBIA\_CAL[0])));

pca.writeMicroseconds(COXA2\_SERVO,angle((90+COXA\_CAL[1])));

pca.writeMicroseconds(FEMUR2\_SERVO,angle((90+FEMUR\_CAL[1])));

pca.writeMicroseconds(TIBIA2\_SERVO,angle((90+TIBIA\_CAL[1])));

pca.writeMicroseconds(COXA3\_SERVO,angle((90+COXA\_CAL[2])));

pca.writeMicroseconds(FEMUR3\_SERVO,angle((90+FEMUR\_CAL[2])));

pca.writeMicroseconds(TIBIA3\_SERVO,angle((90+TIBIA\_CAL[2])));

pca.writeMicroseconds(COXA4\_SERVO,angle((90+COXA\_CAL[3])));

pca.writeMicroseconds(FEMUR4\_SERVO,angle((90+FEMUR\_CAL[3])));

pca.writeMicroseconds(TIBIA4\_SERVO,angle((90+TIBIA\_CAL[3])));

pca.writeMicroseconds(COXA5\_SERVO,angle((90+COXA\_CAL[4])));

pca.writeMicroseconds(FEMUR5\_SERVO,angle((90+FEMUR\_CAL[4])));

pca.writeMicroseconds(TIBIA5\_SERVO,angle((90+TIBIA\_CAL[4])));

coxa6\_servo.write(90+COXA\_CAL[5]);

femur6\_servo.write(90+FEMUR\_CAL[5]);

tibia6\_servo.write(90+TIBIA\_CAL[5]);

}

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

// Print Debug Data

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

void resetData() {

// Set initial default values

data.PSB\_PAD\_DOWN = 1 ;

data.PSB\_PAD\_UP = 1;

data.PSB\_PAD\_RIGHT = 1;

data.PSB\_PAD\_LEFT = 1;

data.PSB\_CIRCLE = 1;

data.PSB\_TRIANGLE =1;

data.PSB\_CROSS = 1;

data.PSB\_SQUARE = 1;

data.PSB\_SELECT = 1;

data.PSB\_START = 1;

data.PSB\_L1 = 1;

data.PSB\_L2 = 1;

data.PSS\_RX = 0;

data.PSS\_RY = 0;

data.PSS\_LX = 0;

data.PSS\_LY = 0;

}

int angle(int x){

x=map(x,0,180,610,2400);

return x;

}