Notes for Haply project

When using the device, the switch on the encoder (the chip at the back of the motor) needs to be on the left side to be on, so the handle on the screen will be able to move when you move the handle.

import processing.serial.\*;

//====spring======

class Spring {

// Screen values

float xpos, ypos;

float tempxpos, tempypos;

int size = 20;

boolean over = false;

boolean move = false;

// Spring simulation constants

float mass; // Mass

float k = 0.2; // Spring constant

float damp; // Damping

float rest\_posx; // Rest position X

float rest\_posy; // Rest position Y

// Spring simulation variables

//float pos = 20.0; // Position

float velx = 0.0; // X Velocity

float vely = 0.0; // Y Velocity

float accel = 0; // Acceleration

float force = 0; // Force

Spring[] friends;

int me;

// Constructor

Spring(float x, float y, float s, float d, float m,

float k\_in, Spring[] others, int id) {

xpos = tempxpos = x;

ypos = tempypos = y;

rest\_posx = x;

rest\_posy = y;

size = (int)s;

damp = d;

mass = m;

k = k\_in; //spring constant

friends = others;

me = id;

}

void update() {

if (move) {

rest\_posy = mouseY;

rest\_posx = mouseX;

}

force = -k \* (tempypos - rest\_posy); // f=-ky

accel = force / mass; // Set the acceleration, f=ma == a=f/m

vely = damp \* (vely + accel); // Set the velocity

tempypos = tempypos + vely; // Updated position

force = -k \* (tempxpos - rest\_posx); // f=-ky

accel = force / mass; // Set the acceleration, f=ma == a=f/m

velx = damp \* (velx + accel); // Set the velocity

tempxpos = tempxpos + velx; // Updated position

if ((overEvent() || move) && !otherOver() ) {

over = true;

} else {

over = false;

}

}

// Test to see if mouse is over this spring

boolean overEvent() {

float disX = tempxpos - mouseX;

float disY = tempypos - mouseY;

if (sqrt(sq(disX) + sq(disY)) < size/2 ) {

return true;

} else {

return false;

}

}

// Make sure no other springs are active

boolean otherOver() {

for (int i=0; i<num; i++) {

if (i != me) {

if (friends[i].over == true) {

return true;

}

}

}

return false;

}

void display() {

fill(7,80,175);

ellipse(tempxpos, tempypos, size, size);

}

void pressed() {

if (over) {

move = true;

} else {

move = false;

}

}

void released() {

move = false;

rest\_posx = xpos;

rest\_posy = ypos;

}

}

//========endOfSpring==================

// Reference from HelloBall (may need to copy other files from HelloBall as well

Device haply\_2DoF;

byte deviceID = 5;

Board haply\_board;

DeviceType degreesOfFreedom;

/\* Graphics Parameters \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int baseFrameRate = 500;

long animation\_count = 0;

long haptics\_count = 0;

long count = 0;

PShape pantograph, joint1, joint2, handle;

Spring Earth, Mars, Mercury; //add more later, these are for testing

int l = 2\*50;

int L = 2\*70;

int d = 2\*20;

int r\_ee = d/3; //radius of handle

float r\_Earth = d/2;

float r\_Mars = (d/2) \* 0.532;

float r\_Mercury = (d/2) \* 0.383;

PVector device\_origin = new PVector (0, 0) ;

//dynamics of planets

long oldTimer = 0;

int m\_Earth = 5; //grams may change

int k\_Earth = 2000; //grams/s^2 may change

float pen\_Earth = 0.0; // mm

float b\_air = .2; // grams/s

//PVector f\_gravity = new PVector(0, -8000); // mm/s^2 setting in space

int m\_Mars = 5; //grams may change

int k\_Mars = 2000; //grams/s^2 may change

float pen\_Mars = 0.0; // mm

int m\_Mercury = 5; //grams may change

int k\_Mercury = 2000; //grams/s^2 may change

float pen\_Mercury = 0.0; // mm

//Initial Conditions

PVector pos\_Earth = new PVector(-50,100); // mm

PVector vel\_Earth = new PVector(0,0); // mm/s

PVector f\_Earth = new PVector(0,0); // uN

PVector pos\_Mars = new PVector(50,100); // mm

PVector vel\_Mars = new PVector(0,0); // mm/s

PVector f\_Mars = new PVector(0,0); // uN

PVector pos\_Mercury = new PVector(75,100); // mm

PVector vel\_Mercury = new PVector(0,0); // mm/s

PVector f\_Mercury = new PVector(0,0); // uN

PVector f\_dampingEarth = new PVector(0,0);

PVector f\_contactEarth = new PVector(0,0);

PVector f\_dampingMars = new PVector(0,0);

PVector f\_contactMars = new PVector(0,0);

PVector f\_dampingMercury = new PVector(0,0);

PVector f\_contactMercury = new PVector(0,0);

// generic data for a 2DOF device

// joint space

PVector angles = new PVector(0,0);

PVector torques = new PVector(0,0);

//task space

PVector pos\_ee = new PVector(0,0);

PVector f\_eeEarth = new PVector(0,0);

PVector f\_eeMars = new PVector(0,0);

PVector f\_eeMercury = new PVector(0,0);

//======spring=======

int num = 3;

Spring[] springs = new Spring[num];

void setup() {

/\*Setup for the graphic display window and drawing objects\*/

size(600, 400, P2D);

background(255);

frameRate(baseFrameRate);

/\* Initialization of the Board, Device, and Device Components\*/

//BOARD

haply\_board = new Board(Serial.list()[0], 0);

//DEVICE

haply\_2DoF = new Device(degreesOfFreedom.TwoDOF, deviceID, haply\_board);

haply\_2DoF.device\_set\_parameters();

device\_origin.add(width/2, height/4 );

createpantograph();

}

/\*\*

\* @brief Main draw function, updates frame at perscribed frame rate

\*/

void draw(){

count = millis();

scale(1,-1);

translate(0,-height);

if(haply\_board.data\_available()){

/\*\*\* GET END-EFFECTOR POSITION (TASK SPACE)\*\*\*\*/

haply\_2DoF.device\_read\_angles();

/\* forward kinematics calculation \*/

angles.x = haply\_2DoF.encoders[0].get\_angle();

angles.y = haply\_2DoF.encoders[1].get\_angle();

haply\_2DoF.mechanisms.forwardKinematics(angles.array());

pos\_ee.set( haply\_2DoF.mechanisms.get\_coordinate());

/\*\*\* PHYSICS OF THE SIMULATION \*\*\*\*/

//Contact Forces

PVector vec\_ee2Earth = (pos\_Earth.copy()).sub(pos\_ee);

PVector vec\_ee2Mars = (pos\_Mars.copy()).sub(pos\_ee);

PVector vec\_ee2Mercury = (pos\_Mercury.copy()).sub(pos\_ee);

float vec\_ee2Earth\_magnitude = vec\_ee2Earth.mag();

float vec\_ee2Mars\_magnitude = vec\_ee2Mars.mag();

float vec\_ee2Mercury\_magnitude = vec\_ee2Mercury.mag();

pen\_Earth = vec\_ee2Earth\_magnitude - (r\_Earth+r\_ee);

pen\_Mars = vec\_ee2Mars\_magnitude - (r\_Mars+r\_ee);

pen\_Mercury = vec\_ee2Mercury\_magnitude - (r\_Mercury+r\_ee);

// planet forces

//Earth

if(pen\_Earth<0){

f\_contactEarth= vec\_ee2Earth.normalize();

f\_contactEarth= f\_contactEarth.mult(-k\_Earth\*pen\_Earth); // since pen\_Earth is negative k\_ball must be negative to ensure the force acts along the end-effector to the ball

}

else{

f\_contactEarth.set(0,0);

}

//Mars

if(pen\_Mars<0){

f\_contactMars= vec\_ee2Mars.normalize();

f\_contactMars= f\_contactMars.mult(-k\_Mars\*pen\_Mars); // since pen\_ball is negative k\_ball must be negative to ensure the force acts along the end-effector to the ball

}

else{

f\_contactMars.set(0,0);

}

//Mercury

if(pen\_Mercury<0){

f\_contactMercury= vec\_ee2Mercury.normalize();

f\_contactMercury= f\_contactMercury.mult(-k\_Mercury\*pen\_Mercury); // since pen\_ball is negative k\_ball must be negative to ensure the force acts along the end-effector to the ball

}

else{

f\_contactMercury.set(0,0);

}

// forces due to damping

f\_dampingEarth = (vel\_Earth.copy()).mult(-b\_air);

f\_dampingMars = (vel\_Mars.copy()).mult(-b\_air);

f\_dampingMercury = (vel\_Mercury.copy()).mult(-b\_air);

// sum of forces

f\_Earth = (f\_contactEarth.copy()).add(f\_dampingEarth);

f\_eeEarth = (f\_contactEarth.copy()).mult(-1);

f\_Mars = (f\_contactMars.copy()).add(f\_dampingMars);

f\_eeMars = (f\_contactMars.copy()).mult(-1);

f\_Mercury = (f\_contactMercury.copy()).add(f\_dampingMercury);

f\_eeMercury = (f\_contactMercury.copy()).mult(-1);

haply\_2DoF.mechanisms.torqueCalculation(f\_eeEarth.array());

haply\_2DoF.mechanisms.torqueCalculation(f\_eeMars.array());

haply\_2DoF.mechanisms.torqueCalculation(f\_eeMercury.array());

torques.set(haply\_2DoF.mechanisms.get\_torque());

haply\_2DoF.motors[0].set\_torque(torques.x);

haply\_2DoF.motors[1].set\_torque(torques.y);

for (Spring spring : springs) {

spring.update();

spring.display();

}

// INTEGRATE THE ACCELERATION TO GET THE STATES OF THE BALL

long currentTimer = count;

float dt = (float)(currentTimer - oldTimer);

println(dt);

//dt = dt/1000;

//dt = (dt < 0.001 )? 0.002 : dt;

//println(dt);

dt=.002;

//println(dt);

pos\_Earth = (((f\_Earth.copy()).div(2\*m\_Earth)).mult(dt\*dt)).add(((vel\_Earth.copy()).mult(dt))).add(pos\_Earth);

vel\_Earth = (((f\_Earth.copy()).div(m\_Earth)).mult(dt)).add(vel\_Earth);

pos\_Mars = (((f\_Mars.copy()).div(2\*m\_Mars)).mult(dt\*dt)).add(((vel\_Mars.copy()).mult(dt))).add(pos\_Mars);

vel\_Mars = (((f\_Mars.copy()).div(m\_Mars)).mult(dt)).add(vel\_Mars);

pos\_Mercury = (((f\_Mercury.copy()).div(2\*m\_Mercury)).mult(dt\*dt)).add(((vel\_Mercury.copy()).mult(dt))).add(pos\_Mercury);

vel\_Mercury = (((f\_Mercury.copy()).div(m\_Mercury)).mult(dt)).add(vel\_Mercury);

oldTimer = currentTimer;

}

/\*\*\*\*\*\*\* ANIMATION TIMER \*\*\*\*\*\*\*\*/

if((count-animation\_count) > 16){

angles.set(haply\_2DoF.mechanisms.get\_angle());

pos\_ee.set(haply\_2DoF.mechanisms.get\_coordinate());

update\_animation(angles.x, angles.y, pos\_ee.x, pos\_ee.y);

}

/\*\*\*\*\*\*\*\*\*\* HAPTICS TIMER \*\*\*\*\*\*\*\*\*\*\*\*\*/

if((count - haptics\_count) > 1){

haply\_2DoF.device\_write\_torques();

}

}

/\* Graphical and physics functions -----------------------------------------------------\*/

/\*\*

\* @brief Specifies the parameters for a haply\_2DoF pantograph animation

\* @note Currently under prototype

\* @param None

\* @return None

\*/

void createpantograph(){

pantograph = createShape();

pantograph.beginShape();

pantograph.fill(0);

pantograph.stroke(255);

pantograph.strokeWeight(2);

pantograph.vertex(device\_origin.x, device\_origin.y);

pantograph.vertex(device\_origin.x, device\_origin.y);

pantograph.vertex(device\_origin.x, device\_origin.y);

pantograph.vertex(device\_origin.x+d, device\_origin.y);

pantograph.vertex(device\_origin.x+d, device\_origin.y);

pantograph.endShape(CLOSE);

joint1 = createShape(ELLIPSE, device\_origin.x, device\_origin.y, d/5, d/5);

joint1.setStroke(color(255));

joint2 = createShape(ELLIPSE, device\_origin.x+d, device\_origin.y, d/5, d/5);

joint2.setStroke(color(255));

handle = createShape(ELLIPSE, device\_origin.x, device\_origin.y, 2\*r\_ee, 2\*r\_ee);

handle.setStroke(color(255));

//Earth = createShape(ELLIPSE, device\_origin.x, device\_origin.y, 2\*r\_Earth, 2\*r\_Earth);

//Earth.setStroke(color(255));

Earth = new Spring(device\_origin.x, device\_origin.y, 2\*r\_Earth, 10.0, 10.0, 0.1, springs, 0);

springs[0] = Earth;

//Earth.setFill(color(7,80,175));

//Mars = createShape(ELLIPSE, device\_origin.x, device\_origin.y, 2\*r\_Mars, 2\*r\_Mars);

//Mars.setStroke(color(255));

Mars = new Spring(device\_origin.x, device\_origin.y, 2\*r\_Mars, 10.0, 10\*0.107, 0.1, springs, 1);

springs[1] = Mars;

//Mars.setFill(color(196, 69, 27));

//Mercury = createShape(ELLIPSE, device\_origin.x, device\_origin.y, 2\*r\_Mercury, 2\*r\_Mercury);

//Mercury.setStroke(color(255));

Mercury = new Spring(device\_origin.x, device\_origin.y, 2\*r\_Mercury, 10.0, 10\*0.0553, 0.1, springs, 2);

springs[2] = Mercury;

//Mercury.setFill(color(219, 209, 206));

}

void update\_animation(float th1, float th2, float x\_E, float y\_E){

background(0); // To clean up the left-overs of drawings from the previous loop!

pantograph.setVertex(1,device\_origin.x+l\*cos(th1), device\_origin.y+l\*sin(th1)); // Vertex A with th1 from encoder reading

pantograph.setVertex(3,device\_origin.x+d+l\*cos(th2), device\_origin.y+l\*sin(th2)); // Vertex B with th2 from encoder reading

pantograph.setVertex(2,device\_origin.x+x\_E, device\_origin.y+y\_E); // Vertex E from Fwd Kin calculations

shape(pantograph); // Display the pantograph

shape(joint1);

shape(joint2);

shape(handle,x\_E, y\_E);

stroke(255);

//shape(Earth, pos\_Earth.x, pos\_Earth.y);

//stroke(255);

//shape(Mars, pos\_Mars.x, pos\_Mars.y);

//stroke(255);

//shape(Mercury, pos\_Mercury.x, pos\_Mercury.y);

//stroke(255);

}

////ENCODERS

//haply\_2DoF.set\_encoder\_parameters(1, 180, 13824, 1);

//haply\_2DoF.set\_encoder\_parameters(2, 0, 13824, 2);

////MOTORS

//haply\_2DoF.set\_actuator\_parameters(1, 1);

//haply\_2DoF.set\_actuator\_parameters(2, 2);

//MECHANISM

//haply\_2DoF.set\_new\_mechanism(NewMech);

//float[] parameters = {(float)l, (float)L, (float)d, 2000 }; //device link parameters

//haply\_2DoF.mechanisms.set\_mechanism\_parameters(parameters);