

ME-GY 6933 ADVANCED MECHATRONICS

Term Project Report

Team Members:

Akshay Kumar V Kutty (avk322)

Christian Lourido (cl4906)

Jordan Birnbaum (jbb498)

Instructor:

Prof Vikram Kapila



NYU

TANDON SCHOOL
OF ENGINEERING

Project Details-

STUDENT NAME:	1. Akshay Kumar V Kutty 2. Christian Lourido 3. Jordan Birnbaum
STUDENT NUMBER:	1. N18887011 2. N13153509 3. N10067813
COURSE NAME:	Advance Mechatronics
DEPARTMENT:	Department of Mechanical and Aerospace Engineering
COURSE CODE:	ME-GY 6933
SUPERVISOR:	Vikram Kapila
DATE OF SUBMISSION:	05-16-2019

TABLE OF CONTENTS

SL NO.	CONTENT	PAGE NO.
1.	OVERVIEW	4
1.1	GOAL	4
1.2	OBJECTIVE OF THE PROJECT	4
1.3	MEASUREMENT	4
2.	INTRODUCTION	5
2.1	BACKGROUND AND SIGNIFICANCE	5
3.	DEVELOPMENT	5
3.1	METHODOLOGY	5
3.2	MECHANICAL DESIGN AND FABRICATION	5
3.3	ELECTRONIC CIRCUIT DESIGN	8
3.4	SELECTION OF COMPONENTS	9
3.5	SOFTWARE	10
4.	BUDGET	11
5.	CONCLUSION AND RESULTS	12
6.	APPLICATIONS	13
7.	FUTURE WORK	13
8.	REFERENCES	13
9.	APPENDIX	14
9.1	APPENDIX -I : EXPERIMENTS AND CALCULATION	14
9.2	APPENDIX -II : CODE	17
9.3	APPENDIX -III : MODULES	31

1. OVERVIEW

1.1 GOAL

The goal of the project includes:

- To build a small scale size of an automated construction crane
- Opensource platform for engineers willing to test and verify simulations and algorithms
- Having a friendly user-interface.

1.2 OBJECTIVE OF THE PROJECT

The objectives for the project includes:

- To build a small scaled module of a tower crane
- Able to move the crane using actuators
- A defined workspace and a processing station
- Develop a control algorithm using serial manipulator kinematics
- Having a friendly user interface
- Manual operation to algorithms for autonomous operation
- Allows the users to point out the desired location visually

the following parameters will be met in establishing and implementing the processes:

- Making the hardware and building it in makerspace
- Developing a control algorithm using serial manipulator kinematics
- Having feedback system for all the actuators
- Pointing out the coordinates using OpenCV

1.3 MEASUREMENT

The success in meeting the objectives would be measured by factors which include:

- Moving all the actuations of the crane manually
- Performing both Forward and inverse kinematics
- Pointing out the exact location using OpenCV camera and moving the crane to that exact position

2. INTRODUCTION

2.1 BACKGROUND AND SIGNIFICANCE

Despite their key role in determining the efficiency of many operations at construction sites, tower cranes have not substantially developed for quite a while. It appears that most users enjoy the essential conveying convenience offered by cranes, and do not bother with "minor" improvements of their performance. Nevertheless, today's technologies in the fields of Robotics and Automation permit far reaching improvements in tower cranes as well. Within this project we have built a small scaled tower crane for educational platform which can be controlled both manually and controlled using and interface by the means of OpenCV.

3. DEVELOPMENT

3.1 METHODOLOGY

The plan is to divide the project in 3 stages:

Phase A (Arduino Based)

- Build the tower crane structure
- Control the actuators manually

Phase B (Propeller Based)

- Automation of sequential tasks
- Establish a workspace
- Programming of functions to allow picking and leaving objects in specific points in the workspace.
- Performing both forward and inverse kinematics

Phase C (Raspberry Pi + Propeller Based)

- Point locations with the means of cameras
- Programming of functions to allow the crane to go to that particular position pinpointed by the camera

3.2 MECHANICAL DESIGN AND FABRICATION

The design of whole structure was laser cut and assembled at makerspace after various iterations in the designs of the mount for the three actuators.

3.2.1 LASER CUT SETTINGS

- MACHINE-Epilog Mini II
- WORKSPACE-24" x 12" (610 x 305 mm)
- Acrylic sheet 1/8" [Speed- 20, Power- 100]
- Birch wood 1/8" [Speed- 40, Power- 100]

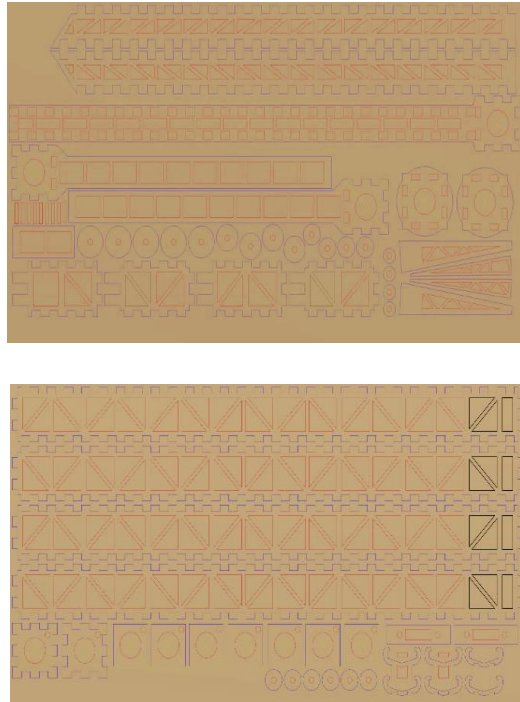


Fig a, b: Tower crane Structure

3.2.2 VARIOUS CHALLENGES FACED IN THE DESIGNS AND FABRICATIONS

- Structure of tower crane- Multiple design iterations
- Carriage mechanism- Tension driven transmission of carriage
- Carriage motor failed after designing entire mount and transmission
- The Servo motor was not stable and would make the crane unstable-Secured servo mount to neck of the crane
- The feedback for the carriage was fixed by designing and 3D printed custom mount for ultrasonic sensor
- Feedback for the gripper motor-Implemented a limit switch
- The neck of the crane-Improved structural integrity
- Defining home pose of crane-Initial Position of the crane when powering up

These are the Solidworks designs of various mounts and parts that had to be 3D printed.

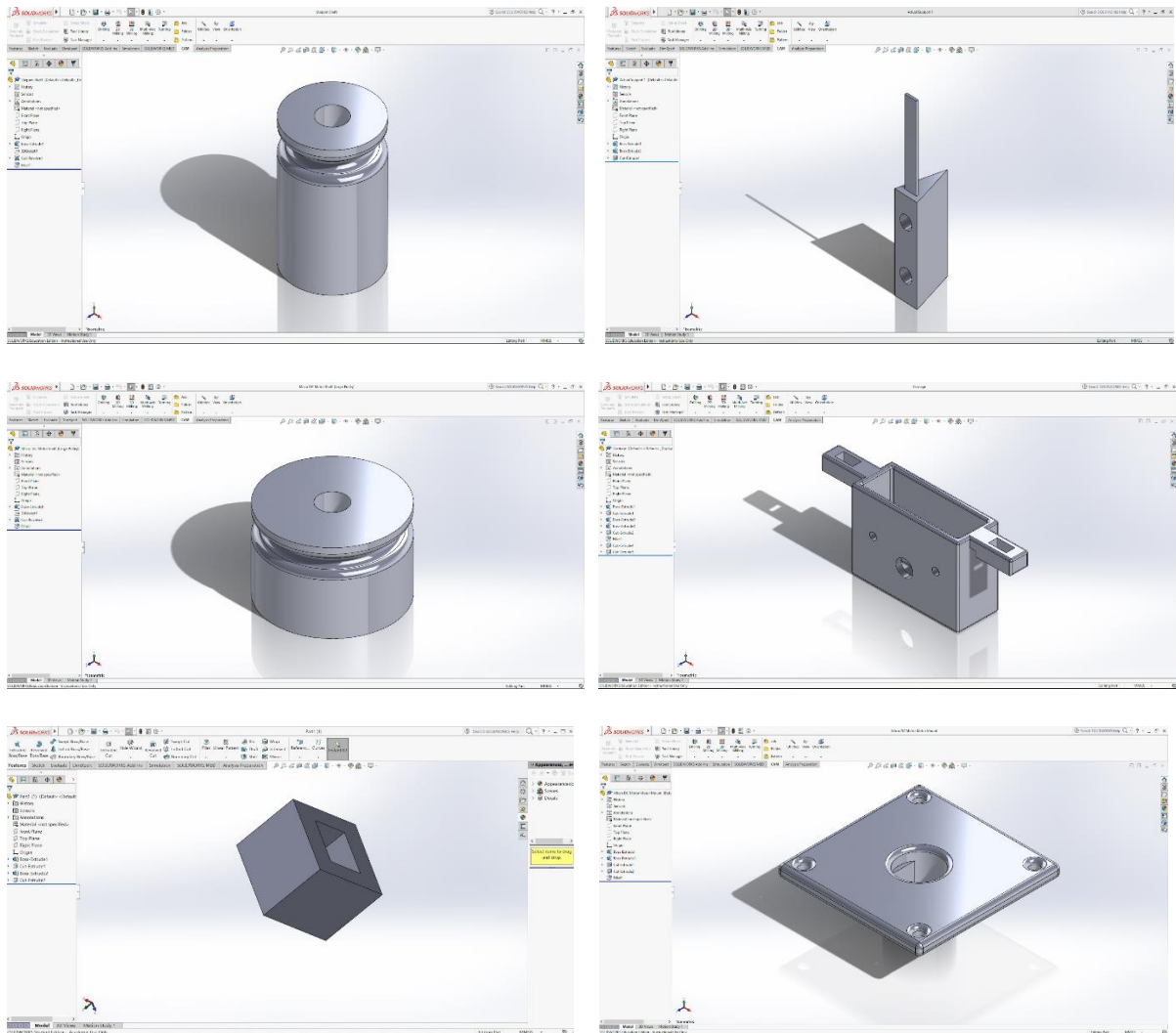


Fig a, b, c, d, e, f: Solidworks Design of various parts

3.2.3 Forward Kinematics

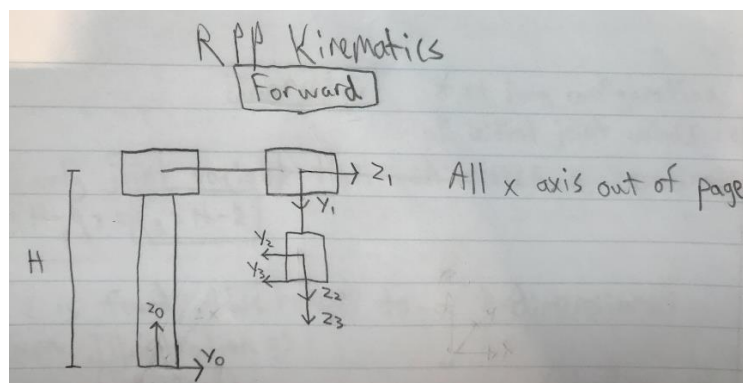


Fig: Assigning the Frames FBD

- Defined home configuration and obtained DH parameters
- Created DH matrices from DH parameters
- Obtained descriptor matrix from end effector to global frame

Joint #	Theta(i)	d(i)	a(i)	alpha(i)	q(i)
1	theta1=-90+q1	H	0	-90	q1
2	0	d2=15+q2	0	-90	q2
3	0	d3=9+q3	0	0	q3

3.2.4 Inverse Kinematics

Note-At home configuration all initial joint values are zero

Q3 is only joint variable that can affect z coordinate

$$Z = H - q$$

$$q3 = H - z$$

Once z is found, this reduces to a 2 dimensional problem (location of carriage)

Schematically,

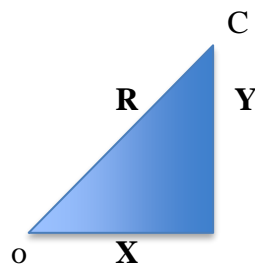


Fig: C= Carriage Location

q2 is only joint variables that can effect R distance ,

|R|=q2, solve for R

$$q2 = \sqrt{x^2 + y^2}$$

q1 is determined by taking the inverse tangent of the x and y values, the atan2 function is used to determine the precise q1 value.

$$\tan Q1 = \frac{y}{x}$$

$$q1 = \text{atan2}(y, x)$$

3.3 ELECTRONICS CIRCUIT DESIGN

The electronic circuit was divided into three circuits:

- Propeller Control Circuit
- Raspberry Pi Control Circuit
- Actuator Circuit

All these circuits are separated and have a common ground. There is one switch which power ups the actuators.

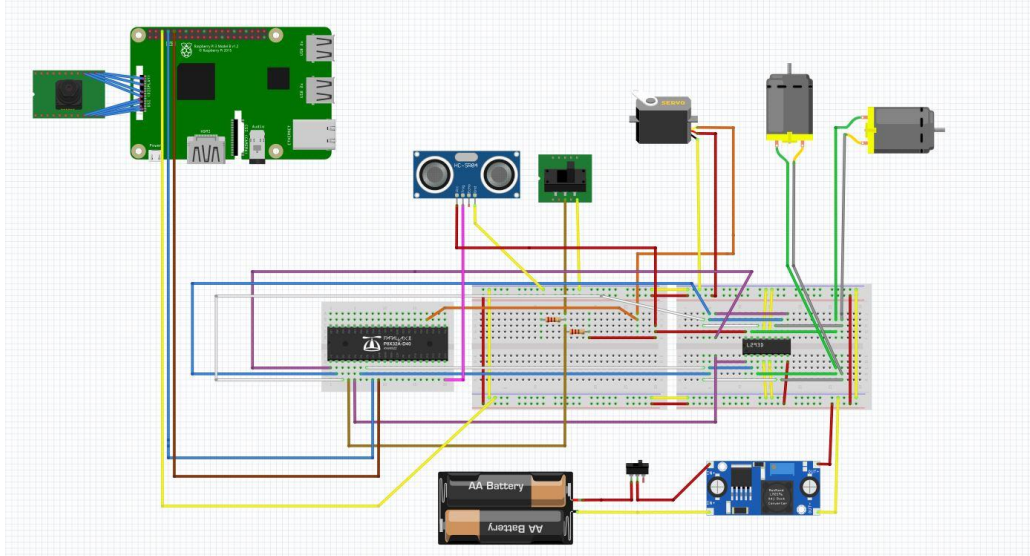


Fig: Electrical Circuit Fritzing

3.4 SELECTION OF COMPONENTS

Raspberry Pi 3 Single Board Computer

Raspberry Pi 3 was selected to handle image processing and send the coordinates once the user clicks the desired location. Since Raspberry Pi 3 is a single board computer with 1 GB of RAM, it will have enough capacity to handle image processing, which demands many processing powers.

Pi Camera

This component was selected because of the friendly compatibility with Raspberry Pi 3 and low cost.

Propeller BoE (Microcontroller)

Since all the controlling are done using the Propeller as it is multicore and cane perform tasks simultaneously. The propeller receives the coordinates from the raspberry pi and performs the inverse kinematics algorithm to move to the desired location.

Ultrasonic Sensors- Ping (One)

Ultrasonic sensors were used to measure distance of the carriage from the initial home position. These were used to act a feedback for the second motor. They were selected because their low cost.

Servo Motor

Used to move the crane (acts as a revolute joint).

DC Motor (12v, 21Rpm)

Used to control the carriage and gripper movement of the crane.

Batteries (3.7V) – 3000 mAh (Power supply for the entire circuit)

Because of the current needed, 2 batteries connected in series (7.4 V total) and 3000mAh is enough to supply power to the actuator circuit.

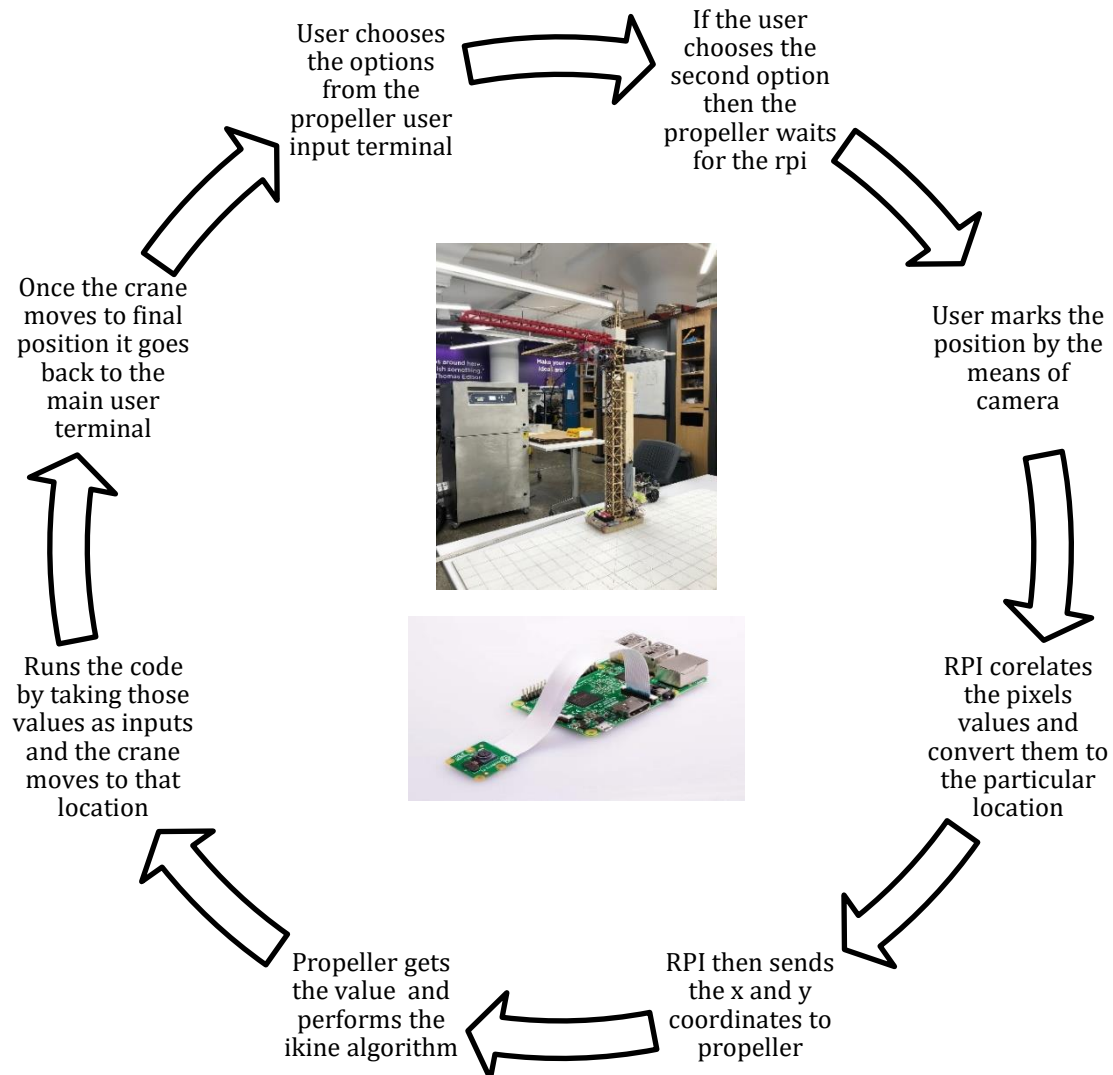
DC-DC Step Down 5V – 2.5 A

Needed to conditionate voltage from the batteries (7.4 V) to 5 V, that is the supply power needed by the servo motor to operate and the voltage needed to activate the relay

ON/OFF switch

Opens the circuit that supply power to the actuator circuit

3.5 SOFTWARE



4. BUDGET

This is the overall cost to build and fabricate the tower crane. The cost of building can be reduced if mass produced for educational purposes.

Bills Of Materials(Prototype)			
Qty	Components	Price	Total
1	Raspberry Pi 3 B+	\$ 30	\$ 30
1	Raspberry Pi Cam V2.1	\$18	\$18
1	Servo Motor	\$12	\$12
1	Dc Motor for the carriage	\$20	\$20
1	Dc Motor for the gripper	\$18	\$18
1	Limit Switch	\$4	\$4
1	Voltage Regulator	\$9	\$9
2	Acrylic Sheet (24*12)	\$15	\$30
1	Motor Driver-L293D	\$ 5	\$ 5
4	Birch Wood (24*12)	\$15	\$60
1	Gorilla glue	\$10	\$10
1	Wood beams	\$9	\$9
1	Corner and L-clamps	\$14	\$14
1	Ping Sensor	\$30	\$30
1	Battery 7.2V	\$17	\$17
1	Breadboard	\$6	\$6
1	Wires	Free	Free
1	Fishing Ropes and Tension Wire	\$11	\$11
1	Miscellaneous	\$ 30	\$ 30
	TOTAL AMOUNT	\$273	\$343

Fig: Cost Analysis

5. CONCLUSIONS AND RESULTS

After successfully designing and manufacturing our crane, naturally, the challenge of how to control the structure autonomously came next. Originally we decided upon controlling the structure with Arduino, since at this stage of the project, functionality and effectiveness of the motors were the most sought after result. Once we had validated that our choice of motors and applied loads was achievable, we moved on to developing a more sophisticated control algorithm. The first step was to choose a more advanced microcontroller. We decided upon propeller since the multicore functionality of it served as a reliable choice for controlling the various motors in our system. After modifying our electrical hardware and making improvements in the mechanical realm, we developed a forward kinematics code to control the crane in a method that was more intuitive and user friendly. The final step of our project was to go beyond the envelope of current crane technology and implement a global vision system that had the ability to communicate desired coordinates to the established system. We executed this by creating a tripod in which we mounted a raspberry pi and pi cam to. The raspberry pi processed the image taken by the pi cam. Once the picture was displayed on the terminal, the user was able to click the desired location of where the crane should lower its end effector to. We wrote a program to convert the pixel values obtained by the click to physical world distance. Since we knew the area of our workspace, the conversion code was a simple ratio. Once the click coordinates were converted to physical world distances, we sent the values over to the propeller via a connection we made between the rx and tx pins. We appended some code to decode the incoming data to the propeller. At this point, the propeller solved for the inverse kinematics and moved the motors in order to achieve the desired position. Our final tests with the crane yielded an impressive accuracy rating of within two and a fifth centimetres of the desired location. Given that our predefined margin for error was five centimetres, we consider the final version of our project a huge success.

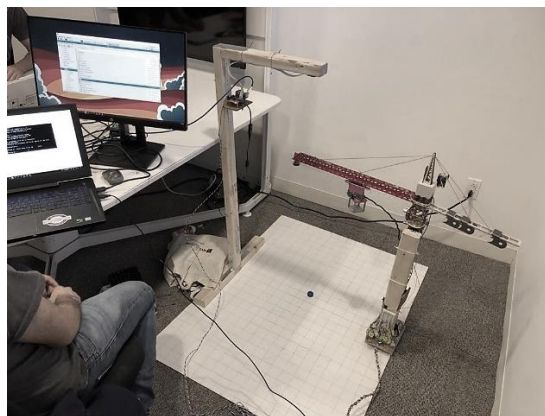


Fig: Experimental Setup

6.APPLICATIONS

1. This can be used for various research simulations and can act as an educational platform for the users
2. This can be used in real life scenarios like construction to reduce the manpower and have less cycling time

7. FUTURE WORK

1. Improve the design and the stability of the tower crane
2. Trying to reduce the error by significant value
3. Have a proper workspace and experimental work
4. Use stereo camera to have better image processing
5. Use of April tags to find the coordinates of a particular location

8. REFERENCES

1. Propeller Advance Mechatronics Class Notes
2. Raspberry pi Advance Mechatronics Class Notes
3. <https://www.teachmemicro.com/raspberry-pi-serial-uart-tutorial/>
4. <https://learn.parallax.com/tutorials/language/propeller-c/propeller-c-set-simpleide>
5. <http://opencvexamples.blogspot.com/2014/01/detect-mouse-clicks-and-moves-on-image.html>
6. https://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html?highlight=setmousecallback

APPENDIX-I: EXPERIMENTATIONS AND CALCULATIONS

FORWARD KINEMATICS-ERROR DIFFERENCE

q1=135 deg, q2= 30, q3=55

Inputs	X Dist.	Y Dist.	Z Dist.	Avg Diff.
Comp Result	-21	21	-2	-
Physical Result	-18	23	5	2.67
	-18.5	22.2	6	1.57
	-18.5	22.5	3.5	3.83

q1=30 deg, q2= 35, q3=30

Inputs	X Dist.	Y Dist.	Z Dist.	Avg Diff.
Comp Result	30.3	17.5	28	-
Physical Result	23.5	18	30	4.13
	29	20	31	1.5
	26	19	31	2.67

q1=0 deg, q2= 25, q3=40

Inputs	X Dist.	Y Dist.	Z Dist.	Avg Diff.
Comp Result	25	0	18	-
Physical Result	22	2	22	1.83
	26	3	22	2.67
	26.5	3	21	3.67

Average Difference for all the trials:- x:2.88, y:1.91, z:3.39

WITHOUT CALIBRATION ALGORITHM-ERROR DIFFERENCE

TRIAL-I

Inputs	I-click	II-click	III-click	Avg Diff.
Clicked	(-20,15)	(-20,15)	(-20,15)	-
Computed	(-21,16)	(-21,16)	(-21,16)	1.1
Placed	(-17.5,15)	(-17.5,15)	(-17.5,15)	(3,1.16)
Z-comp	5cm	5cm	5cm	-
Z-physical	5cm	10cm	5.5cm	1.83

TRIAL-II

Inputs	I-click	II-click	III-click	Avg Diff.
Clicked	(25,30)	(25,30)	(25,30)	-
Computed	(25,32)	(25,32)	(25,32)	0.2
Placed	(20,27.5)	(20,27.5)	(23,26)	(4,3)
Z-comp	5cm	5cm	5cm	-
Z-physical	7.5cm	7cm	7cm	2.167

TRIAL-III

Inputs	I-click	II-click	III-click	Avg Diff.
Clicked	(0,40)	(0,40)	(0,40)	-
Computed	(0,42)	(0,42)	(0,42)	0.2
Placed	(0,36)	(0,36)	(0,35)	(0,4.3)

Z-comp	5cm	5cm	5cm	-
Z-physical	5cm	5cm	4cm	0.33

Average computational difference:- (0.33,1.67) ~1

Average physical difference:- (2.33,2.83) ~2.58

Average total difference:- (1.33,2.25) ~1.79

Average total Z difference :- 1.44

Average spatial difference:- 2.2

APPENDIX- II: CODE

```
/*
MainProgram.c
ME GY-6933 Advanced Mechatronics
Autonomous Crane for Education
Instructor: Professor Vikram Kapila
Date:05/16/2019
Group Members: Christian Lourido(cl4906), Akshay Kumar Kutty(avk322), Jordan
Birnbaum(jbb498)
*/
#include "HeaderFile.h"
int main()
{
int iuAngle,q1new,q2new,q3new, q1,d2,d3,timeq3,xu,yu,zu;
char ui;
double xnew,ynew,znew;

//-----Start COG for PING sensor-----//
cogstart(&pingfromcog, NULL, stack1, sizeof(stack1)); //Start ping sensor in Cog 1
pause(500);

//-----Calibration-----//
print("Calibration...\n\nEnter current position of q1 (deg): "); //Since there is no
feedback, ui is needed as initial reference
scanf(" %d", &iuAngle);
home(iuAngle); //argument is the current angle of q1 before going home

//Initialization of DOFs (values after calibration):
q1old= iPosq1;
q3old= iPosq3;

while(1){
print("Welcome to Crane-0\n\nMain Menu:\n1. I-Kine Mode (Inputs: x, y, z)\n2. Camera
Mode(Select Point in plane)\n3. Return to Home Configuration\n\nOption: ");
scanf(" %c",&ui);
pause(500);

switch(ui){

case '1':
print("\nI-Kine Mode!\n\nEnter the value of x(cms): ");
scanf(" %d",&xu);
print("\nEnter the value of y (cms): ");
```

```

scanf(" %d",&yu);
print("\nEnter the value of z (cms): ");
scanf(" %d",&zu);
ikine(xu,yu,zu);
break;

case '2':
print("Continue to data pulling? (y/n)");
scanf(" %c",&ui);

if(ui=='y'){
unsigned char data_received[20]={};

//Open serial communication
rpi = fdserial_open(9, 8, 0, 115200);
print("Run the executable file in the RPI. \nWaiting for RPI to send data...\n");
dprint(rpi, "\nHi from propeller!"); //To verify in RPI terminal

//Receive array of [20]
for(int i=0; i<20; i++){
data_received[i]=fdserial_rxChar(rpi);
}
pause(1000);

//Data processing
/*Count digits until special character ":" is found. 1st value = X, 2nd value = Y*/
int n=0, digitx=0, digity=0;
while(data_received[n]!=':'){
digitx++;
n++;
}
digitx=n;
n++;
while(data_received[n]!=':'){
digity++;
n++;
}

/*Create arrays for X and Y based on the number of digits each has. Then, put data from
buffer to new X and Y arrays*/
char x[digitx-1],y[digity-1];
int m=0, o=0;
while(m<=digitx-1){

```

```

x[m]=data_received[m];
m++;
}
m++;
while((o+m)<=(digitsx+digitsy)){
y[o]=data_received[m+o];
o++;
}

/*Finally, convert strings to integers */
int xint=atoi(x);
int yint=atoi(y);
pause(500);
ikine(xint,yint,53);//change z eventually
fdserial_close(rpi);
}
else{}
break;

case '3':
print("Return to Home Position\n");
home(q1old);
print("OK\n\n");
break;

case 10:
break;

default:
break;
} } }
//Defined Functions

void pingfromcog(void *par1) {
long tEcho;
while(1)
{
tEcho=ping(15);
cmDist = tEcho/58.0; // Get cm distance from Ping)))
pause(20); // Wait 1/5 second
}
}

```

```

void fkine(int q1f, int q1fold, int q2f, int q3f, int q3fold){
/*Move servo 1 to desired position*/
print("\nMoving q1...");
ServoMove(smPin,q1fold,q1f,smSpeed);
print("Final value of q1: %d\tDone!\n", q1f);

/*Move DC Motor 2 to desired position*/
print("\nMoving q2...");
if((totalOffset+cmDist)>q2f){
while((totalOffset+cmDist)>q2f){
dcMove(en2, in21, in22, inside, on);
}
dcMove(en2, in21, in22, outside, off);
}
else if((totalOffset+cmDist)<q2f){
while((totalOffset+cmDist)<q2f){
dcMove(en2, in21, in22, outside, on);
}
dcMove(en2, in21, in22, outside, off);
}
print("Final value of q2: %.2f\t Done!\n", cmDist+totalOffset);
/*Move DC Motor 2 to desired position*/
print("\nMoving q3...");
int time;
time=(q3f*10000/speedConstant);
if (q3f>q3fold){
dcMoveTimed(en3, in31, in32,down,time);
}
else if (q3f<q3fold){
dcMoveTimed(en3, in31, in32,up,time);
}
print("Final value of q3: %d\t Done!\n", q3f);
print("\nDone!\n\n");
}

void home(int initialAngle){
//Servomotor 1: Take servo to initial position
print("Moving q1...\t");
ServoMove(smPin,initialAngle,iPosq1,smSpeed);//Move q1 from UserInput Angle to
iPosq1
print("q1= %d\n",iPosq1);

//DC Motor 2: Take Carriage to home position

```

```

print("Done!\n\nMoving q2...\t");
if(cmDist>iPosq2){
while(cmDist>iPosq2){
dcMove(en2, in21, in22, inside, on); //Turn on motor until carriage reaches initial
position
}
dcMove(en2, in21, in22, inside, off); // STOP MOTOR
}
else if(cmDist<iPosq2){
while(cmDist<=iPosq2){
dcMove(en2, in21, in22, outside, on);
}
dcMove(en2, in21, in22, inside, off); // STOP MOTOR
}
print("q2= %.2f\n", cmDist+totalOffset);

//DC Motor 3: Take DC motor to home position (Until the limit switch changes its state)
print("Done!\n\nMoving q3...\t");
while(input(limitSwitchPin) != 1){
dcMove(en3, in31, in32, up, on); //direction: 1 (up), 0 (down) dcMove(int enPin, int
dir1Pin, int dir2Pin, int direction, int onOff)
}

dcMove(en3, in31, in32, up, off); //STOP MOTOR dcMove(int enPin, int dir1Pin, int
dir2Pin, int direction, int onOff)
print("q3= 0\n");
print("\nDone!\n\n");
}

void ikine(int xi, int yi, int zi){
float q1newi,q2newi,q3newi;
float veryf=sqrt(pow(xi,2)+pow(yi,2));
char ui2;
if(veryf>15.0 && veryf<45.0 && yi>=0 && zi>=1 && zi<=57){
print("\nThe coordinate (%d,%d,%d) is reachable!\nveryf= %f\n",xi,yi,zi,veryf);

// Inverse Kinematics
q1newi= atan2(yi,xi)*180.0/PI;
q2newi= sqrt(pow(xi,2)+pow(yi,2));
q3newi= H-zi;
print("Calculated dof movements: q1= %.2f, q2= %.2f, q3=
%.2f\n\nConfirm?(y/n)",q1newi, q2newi, q3newi);
scanf(" %c",&ui2);

```

```

if(ui2== 'y'){
fkine(q1newi,q1old,q2newi,q3newi,q3old);
q1old=q1newi;
q3old=q3newi;
}
else {
print("\nOperation cancelled... \n\n");
pause(2000);
} }
else{
print("\nThe coordinate (%d,%d,%d) is NOT reachable!\nverify= %f\n",xi,yi,zi,verify);
} }

```

HEADERFILES

```
/* Forward Declarations */
```

```

#include "simpletools.h"
#include "ping.h"
#include "servo.h"
#include "math.h"
#include "fdserial.h"

```

```
/* Forward Declarations */
```

```

void pingfromcog(void *par1); //PING Sensor
void ServoMove(int pin, int iangle, int fangle,int delay); //Servo motor function
void dcMove(int enPin, int dir1Pin, int dir2Pin, int direction, int onOff); //DC motor
control - ON/OFF
void fkine(int q1f,int q1fold, int q2f, int q3f,int q3fold);
void home(int initialAngle);
void ikine(int xi, int yi, int zi);
/*Global variables*/

```

```
int q1old,q3old;
```

```
/* Variables for cogs */
```

```

unsigned int stack1[40 + 15]; // Stack vars for cog1 (PIN Sengor)
static volatile float cmDist;

```

```
/* Constants */
```

```

/*Initial position constants*/

const int iPosq1=0; //Initial value for q1 (degrees)
const int iPosq2=5; //Initial value for q2 (cms)(UltraSonic Distance)
const int iPosq3 = 0; //Initial value for q3 (cms)

/* Forward and Inverse Kinematics */

const int totalOffset =10; //neckdist=5.5 + stringoffset=4.5
const int H=58; //Starting height (cms)

/*Neck servo motor constants*/

const int smPin=19; //Servo motor pin
const int smSpeed=50; //Servo motor speed (how fast the angle change will be)

/*Constants for DC Motors*/

const int on=1;
const int off=0;

/*Carriage motor (motor 2) constants*/
const int en2=0; //Enable pin for DC Motor in L293D
const int in21=1; //Input pin 1 for DC Motor in L293D (H-Bridge)
const int in22=2; //Input pin 2 for DC Motor in L293D (H-Bridge)
const int inside=1; //direction (towards the neck)
const int outside=0; //direction (away from neck)

/*Carriage motor (motor 3) constants*/
const int en3=4; //Enable pin for DC Motor in L293D
const int in31=5; //Input pin 1 for DC Motor in L293D (H-Bridge)
const int in32=6; //Input pin 2 for DC Motor in L293D (H-Bridge)
const int up=1; //direction
const int down=0; //direction
const int speedConstant = 10 ; // [cm/s]
const int limitSwitchPin = 3;

fdserial *rpi;

SERVOMOVE

#include "servo.h"
#include "simpletools.h"
void ServoMove(int pin, int iangle, int fangle,int delay){

```

```

int fangleCorr= 5+fangle*(180.0/225.0);
int iangleCorr= 5+iangle*(180.0/225.0);
if (iangleCorr>fangleCorr){
for(int n=iangleCorr; n>=fangleCorr; n--){
servo_angle(pin,n*10);
pause(delay);
}
}else if(iangleCorr<fangleCorr){
for(int n=iangleCorr; n<=fangleCorr;n++){
servo_angle(pin,n*10);
pause(delay);
}
}else{}
}

```

DCMOVE

```

#include "simpletools.h"
void dcMove(int enPin, int dir1Pin, int dir2Pin, int direction, int onOff){
if(onOff==0){
low(dir1Pin); low(dir2Pin); low(enPin);
}
else{
if(direction == 1){
low(dir1Pin); high(dir2Pin); high(enPin);
}else{
high(dir1Pin); low(dir2Pin); high(enPin);
} } }

```

DCTIMEDMOVE

```

#include "simpletools.h"
void dcMoveTimed(int enPin, int dir1Pin, int dir2Pin, int direction, int time){
if(direction == 1){
low(dir1Pin); high(dir2Pin); high(enPin);
pause(time);
low(dir1Pin); low(dir2Pin); low(enPin);
}else if(direction == 0){
high(dir1Pin); low(dir2Pin); high(enPin);
pause(time);
low(dir1Pin); low(dir2Pin); low(enPin);
} }

```


IMAGE LOCATION -CLICK (RPI)

```
#include <ctime>
#include <iostream>
#include <raspicam/raspicam_cv.h>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>           //Used for the UART
#include <termios.h>        //Used for the UART
using namespace std;
using namespace cv;
Mat image;
int TMx=660; //can change TMx,TMy,XPR,YPR based on calibration
int TMy=28;
int XPR=9.4;
int YPR=9.4;
int Yoffset=5; //cm
int Cx;
int Cy;
int xval;
int yval;

static void on_mouse( int e, int x, int y, int d, void *ptr )
{
    if (e == EVENT_LBUTTONDOWN) {
        int x1=x;
        int y1=y;
        Cx=x1;
        Cy=y1;

        cout << "Pixel Position:x(" << Cx << ")" << endl;
        cout << "Pixel Position:y(" << Cy << ")" << endl;
        sleep(1);
        destroyAllWindows();
    } }

int main()
{
    //Establish Serial Comm

    int uart0_filestream = -1;
```

```

uart0_filestream = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY);
    //Open in non-blocking read/write mode

    if (uart0_filestream == -1)
    {
        //ERROR - CAN'T OPEN SERIAL PORT
        printf("Error - Unable to open UART. Ensure it is not in use by another
application\n");
    }

    struct termios options;
    tcgetattr(uart0_filestream, &options);
    options.c_cflag = B115200 | CS8 | CLOCAL | CREAD;           //<Set baud rate
    options.c_iflag = IGNPAR;
    options.c_oflag = 0;
    options.c_lflag = 0;
    tcflush(uart0_filestream, TCIFLUSH);
    tcsetattr(uart0_filestream, TCSANOW, &options);

//Record mouse click on picture code
Point p;
Mat image = imread("raspicam_cv_imageRevM1.jpg");

//pass a pointer to `p` as parameter
cvSetMouseCallback("window",on_mouse, &p);

//from mouse2pixel example
raspicam::RaspiCam_Cv Camera;
if (!Camera.open()){
    cerr<<"Error opening the camera"<<endl;return -1;
}
sleep(1);

//Start capture
Camera.grab();
Camera.retrieve (image);

//Save image
imwrite("raspicam_cv_imageRevM1.jpg",image);
Camera.release();

//Display all images
namedWindow( "Original");

```

```

moveWindow("Original",0,0);
resizeWindow("Original",200,200);
cv::imshow( "Original", image);
cout<<"Click a point on the desired location"<<endl;

//Function
cvSetMouseCallback("Original", on_mouse, (void*) (&p));
waitKey(15000);

//Processing coordinates function: Convert pointer to integer, transform pixel values
to real world distance values
char response;
xval=((TMx-Cx)/XPR);
yval=((Cy-TMy)/YPR)-Yoffset;
cout << "The equivalent real world coordinates are:" << endl;
sleep(1);
cout << "In x-axis(" << xval<< ")" << endl;
cout << "In y-axis(" << yval << ")" << endl;
sleep(1);
cout << "Do you want to proceed with these coordinates(y/n)?" << endl;
cin>>response;

if (response == 'y'){
    cout << "Sending the coordinates to Propeller" << endl

//----- TX BYTES -----
    char tx_buffer[20];

    sprintf(tx_buffer, "%d:%d:",xval,yval);

    if (uart0_filestream != -1)
    {
        int count = write(uart0_filestream, &tx_buffer[0], 20);
        //Filestream, bytes to write, number of bytes to write
        if (count < 0)
        {
            printf("UART TX error\n");
        } }
    }else if (response == 'n'){
        cout << "No valid input, rerun program" << endl;
        sleep(3);
    return 0;
}

```

OBJECT DETECTION (RPI)

USAGE

python ball_tracking.py --video ball_tracking_example.mp4

python ball_tracking.py

import the necessary packages

from collections import deque

from imutils.video import VideoStream

import numpy as np

import argparse

import cv2

import imutils

import time

construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add_argument("-v", "--video",
 help="path to the (optional) video file")

ap.add_argument("-b", "--buffer", type=int, default=64,
 help="max buffer size")

args = vars(ap.parse_args())

define the lower and upper boundaries of the "green"

ball in the HSV color space, then initialize the

list of tracked points

greenLower = (29, 86, 6)

greenUpper = (64, 255, 255)

pts = deque(maxlen=args["buffer"])

if a video path was not supplied, grab the reference

to the webcam

if not args.get("video", False):

 vs = VideoStream(src=0).start()

otherwise, grab a reference to the video file

else:

 vs = cv2.VideoCapture(args["video"])

allow the camera or video file to warm up

time.sleep(2.0)

keep looping

while True:

```

# grab the current frame
frame = vs.read()

# handle the frame from VideoCapture or VideoStream
frame = frame[1] if args.get("video", False) else frame

# if we are viewing a video and we did not grab a frame,
# then we have reached the end of the video
if frame is None:
    break

# resize the frame, blur it, and convert it to the HSV
# color space
frame = imutils.resize(frame, width=600)
blurred = cv2.GaussianBlur(frame, (11, 11), 0)
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

# construct a mask for the color "green", then perform
# a series of dilations and erosions to remove any small
# blobs left in the mask
mask = cv2.inRange(hsv, greenLower, greenUpper)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)

# find contours in the mask and initialize the current
# (x, y) center of the ball
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                        cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
center = None

# only proceed if at least one contour was found
if len(cnts) > 0:
    # find the largest contour in the mask, then use
    # it to compute the minimum enclosing circle and
    # centroid
    c = max(cnts, key=cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

    # only proceed if the radius meets a minimum size
    if radius > 10:

```

```

        # draw the circle and centroid on the frame,
        # then update the list of tracked points
        cv2.circle(frame, (int(x), int(y)), int(radius),
                    (0, 255, 255), 2)
        cv2.circle(frame, center, 5, (0, 0, 255), -1)

    # update the points queue
    pts.appendleft(center)

    # loop over the set of tracked points
    for i in range(1, len(pts)):
        # if either of the tracked points are None, ignore
        # them
        if pts[i - 1] is None or pts[i] is None:
            continue

        # otherwise, compute the thickness of the line and
        # draw the connecting lines
        thickness = int(np.sqrt(args["buffer"] / float(i + 1)) * 2.5)
        cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)

    # show the frame to our screen
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the 'q' key is pressed, stop the loop
    if key == ord("q"):
        break

    # if we are not using a video file, stop the camera video stream
    if not args.get("video", False):
        vs.stop()

    # otherwise, release the camera
    else:
        vs.release()

    # close all windows
    cv2.destroyAllWindows()

```

APPENDIX-III: MODULES

RASPBERRY PI 3 B+



Fig: Raspberry pi 3 B+ model

The Raspberry Pi 3 Model B+ is the latest product in the Raspberry Pi 3 range.

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

Source: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>

PARALLAX PROPELLER BOE

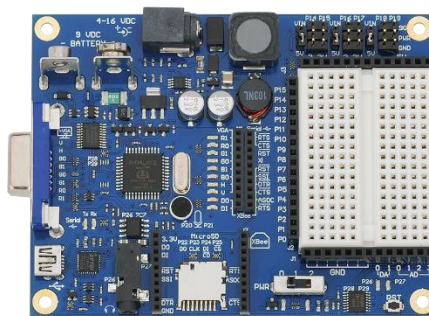


Fig: Propeller Parallax Board of Education

- System clock speed: DC to 80 MHz
 - Global (shared) RAM/ROM: 64 K bytes; 32 K RAM/32 K ROM
 - The HUB manages mutually exclusive resources
- Terminology: HUB --Propeller --Spinner
 - Cog RAM: 512 x 32 bits each, i.e., 512 Longs (or 2KB)

- Last 16 registers (16 longs) are special purpose, dedicated to System Counter, I/O pins, and local cog peripherals
- Power requirements: 4 - 16 VDC
- Communication: USB for programming
- Dimensions: 4.375 x 3.05 x 0.625 in (11.11 x 7.75 x 1.59 cm)
- Operating temp range: +32 to +158 °F (0 to +70 °C)

Source: https://www.parallax.com/sites/default/files/downloads/32900-Propeller-BOE-Documentation-v1.1_0.pdf

PI CAMERA

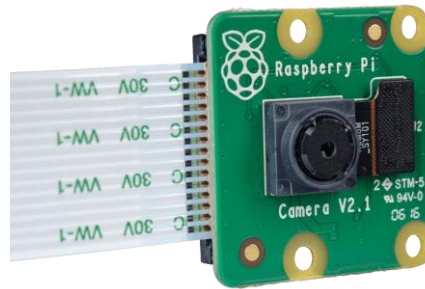


Fig: Raspberry pi Camera V2.1

The Raspberry Pi Camera Board v2 is a high quality 8-megapixel Sony IMX219 image sensor custom designed add-on board for Raspberry Pi, featuring a fixed focus lens. It's capable of 3280 x 2464-pixel static images, and also supports 1080p30, 720p60, and 640x480p90 video.

It attaches to the Pi by way of one of the small sockets on the board's upper surface and uses the dedicated CSI interface, designed especially for interfacing to cameras.

- 8 megapixel camera capable of taking photographs of 3280 x 2464 pixels
- Capture video at 1080p30, 720p60 and 640x480p90 resolutions
- All software is supported within the latest version of Raspbian Operating System
- Applications: CCTV security camera, motion detection, time lapse photography

Source: <https://www.raspberrypi.org/products/camera-module-v2/>

SERVO MOTOR



Fig: Standard Servo motor (Mg995r)

- Weight: 55 g
- Dimension: 40.7×19.7×42.9mm
- Operating voltage range: 4.8 V to 7.2 V
- Stall torque: 9.4kg/cm (4.8v); 11kg/cm (6v)
- Operating speed: 0.2 s/60° (4.8 V), 0.16 s/60° (6 V)
- Rotational degree: 180°
- Dead band width: 5 μ s
- Operating temperature range: 0°C to +55°C
- Current draw at idle: 10mA
- No load operating current draw: 170mA
- Current at maximum load: 1200mA

Source: <https://servodatabase.com/servo/towerpro/mg995>

DC MOTOR



Fig: Dc Motor(12V, 21 RPM)

This is a special Low Speed DC Motor that will spin at a speed of 21 RPM max. It works by having a tiny metal gearbox built-in for durability and speed reduction

- Operating Voltage: 6V - 12 V
- Free-run speed at 12 V: 21 RPM
- Free-run current at 12 V: 80 mA
- Stall current at 12 V: 0.85 A
- Stall torque at 12 V: 32 kg.cm
- Gear ratio: 1:378
- Shaft Type: D Shaft
- Reductor size(L): 27 mm
- Weight: 70 g

Source: <https://tinkersphere.com/motors-wheels/1037-low-speed-dc-motor-12v-21-rpm.html>

DC MOTOR



Fig: Dc Motor(12V, 21 RPM)

Apply 12V DC across the 2 metal connectors at the back of this DC motor to have it spin. This is a special Low Speed DC Motor that will spin at a speed of 21 RPM max. It works by having a tiny metal gearbox built-in for durability and speed reduction.

- Operating Voltage: 12 V
- Free-run speed at 12 V: 21 RPM
- Load speed at 12V: 16 RPM
- Free-run current at 12 V: 60 mA
- Stall torque at 12 V: 0.8 kg.cm 11.1 ozin
- Shaft Size: 5mm
- Total Size(not include shaft): 33*24mm(D*H);
- Material: Plastic
- Weight: 32g

Source: <https://tinkersphere.com/motors-wheels/1151-low-speed-dc-motor-12v-21-rpm-with-plastic-mount.html>

PING SENSOR



Fig: PING Ultrasonic Sensor

Provides precise, non-contact distance measurements within a 3 cm to 3 m range

- Narrow acceptance angle
- Range: approximately 1 inch to 10 feet (3 cm to 3 m)
- 3-pin male header with 0.1" spacing
- Power requirements: +5 VDC; 35 mA active
- Communication: positive TTL pulse
- Dimensions: 0.81 x 1.8 x 0.6 in (22 x 46 x 16 mm)
- Operating temperature range: +32 to +158 °F (0 to +70 °C)

Source: <https://www.parallax.com/product/28015>

MOTOR DRIVER L293D

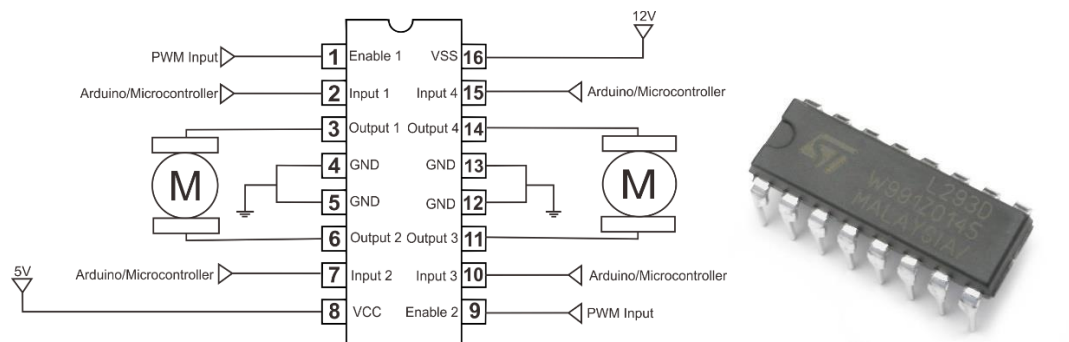


Fig: L293d Motor Driver Chip

The L293 and L293D devices are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. These devices are designed to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

- Wide Supply-Voltage Range: 4.5 V to 36 V
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

Source: <http://www.ti.com/lit/ds/symlink/l293.pdf>

VOLTAGE REGULATOR



Fig: Voltage Regulator

The LM2596 regulator is monolithic integrated circuit ideally suited for easy and convenient design of a step-down switching regulator (buck converter). It is capable of driving a 3.0 A load with excellent line and load regulation

- Adjustable Output Voltage Range 1.23 V – 37 V
- Guaranteed 3.0 A Output Load Current
- Wide Input Voltage Range up to 40 V
- 150 kHz Fixed Frequency Internal Oscillator
- TTL Shutdown Capability
- Low Power Standby Mode, 80 A
- Thermal Shutdown and Current Limit Protection
- Internal Loop Compensation

Source: <https://www.amazon.com/LM2596-Buck-Converter-Adjustable-ModuleVoltage/dp/B01MS4D2FO>

LIMIT SWITCH



Fig: Limit Switch

The limit switch then regulates the electrical circuit that controls the machine and its moving parts. These switches can be used as pilot devices for magnetic starter control circuits, allowing them to start, stop, slow down, or accelerate the functions of an electric motor.

- Max Current: 6A
- Max Voltage: 250VAC
- Mounting Style: 4mm Thread
- Switch Style: Limit Switch
- Throw Persistence: Momentary

Source: <https://www.zoro.com/zoro-select-mini-snap-swch-10a-spdt-roller-lever-6x293/i/G0576922/>