Goal

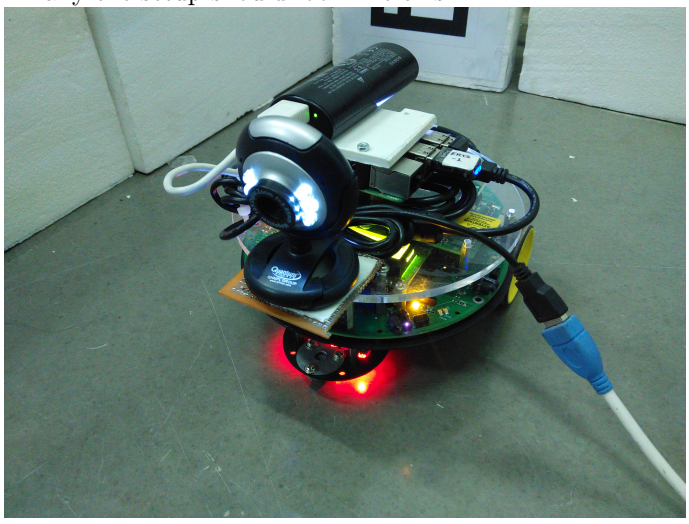
In this chapter:
* We will learn how to detect the marker
* We will learn how to determine the distance, angle of the marker with respect to the camera
* We will learn how to map the real world coordinates on to a virtual arena
* We will learn about socket programming

Pre-requisites

- Aruco markers
- Hamming distance
- Camera Calibration
- Pose estimation
- SSH - MobaXterm

Construction

The Raspberry pi (Raspi) will be mounted on top of the Firebird V robot. To power the Raspi we will use a power bank. For additional power to the Raspi, GPIO pins can be used. An USB webcam will be connected to the USB port of the Raspi.
Finally the setup should look like this:



Things to do:

- Connect the Raspi to the power bank and connect a HDMI cable. Once you have entered the GUI of the Raspi, type `ifconfig` in the terminal and note down the IP address of the network to which the wireless adapter is connected to.
- Open MobaXterm on your PC and enter 'ssh -x pi@192.168.1.119' considering the IP address is 192.168.1.119.
- Once connected you can access the code on the Raspi.

Code

Code to be run on the Raspi. This code will act as the client and send data to the server.

```
      """
     """
Code for localizing a robot with Aruco markers.

Authors: Niharika Jayanthi, Dheeraj Kamath
Project: Marker-based Localization
Mentor: Sanam Shakya

Main functions:

Global variables:

"""

#Imports

import cv2
import socket
import numpy as np
import math



#Global variables

MAX_MARKERS = 4

mark_detect = [0, 0, 0, 0]
```

```
objp = np.zeros((1*4,3), np.float32)

objp[1,1] = 1
objp[2,0] = 1
objp[2,1] = 1
objp[3,0] = 1

objp = objp * 105
#(700/9.605116)

count = 0

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
port = 7000

server_ip = '192.168.1.104'

s.connect((server_ip, port))
print "Connected to server"


#Helper functions

def is_aruco_present(img1):
    """
    * Function Name:    is_aruco_present
    * Input:        Image captured
    * Output:       Returns the set of corner points of the marker in the
                        image.
    * Logic:            Takes the input image and finds the contour having four points and c
    * Example Call: is_aruco_present()
    """

    i = 0
    aruco_points = []
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    ret, thresh1 = cv2.threshold(gray1, 175,255, cv2.THRESH_BINARY_INV+ cv2.THRESH_OTSU)
    img = thresh1.copy()
    kernel = np.ones((5,5), np.uint8)
    open1 = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel)
    median1 = cv2.medianBlur(open1, 5)
    c1, h1 = cv2.findContours(thresh1, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for c in c1:
        k = cv2.isContourConvex(c)      #Check for convexity, removes unwanted curved contou
        if k == True:
            i = i +1
```

```
            continue
        elif k == False:
            e2 = 0.1*cv2.arcLength(c, True)
            a2 = cv2.approxPolyDP(c, e2, True)
            print "a2 = ", a2, "\n", "e2 =", e2
            if len(a2)== 4:
                x = a2[0,0,1]-a2[2,0,1]
                y = a2[2,0,0]-a2[0,0,0]
                if x < 0:
                    x = x * (-1)      #Converting to unsigned int
                if y < 0:
                    y = y * (-1)      #Converting to unsigned int
                if  10 < e2 < 100:              #45-60cm
                    print "Contour Id : ",i,"length of array =", len(a2), "\n""\n", a2
                    print x-y
                    if -50 < x-y < 50:  # Without tilt
                        aruco_points = a2
                        cv2.drawContours(img1, c1, i, (0,255,0), 2)
                        cv2.imshow('img',img1)
                        cv2.waitKey(500)
                        cv2.destroyAllWindows()
                        return aruco_points, '1'  # Flag values can be changed as per conve
                    elif -80 < x-y < 80:  #with tilt
                        aruco_points = a2
                        cv2.drawContours(img1, c1, i, (0,255,0), 2)
                        cv2.imshow('img',img1)
                        cv2.waitKey(500)
                        cv2.destroyAllWindows()
                        return aruco_points, '1'
                elif 100 < x-y < 200:
                    print "Contour Id : ",i,"length of array =", len(a2), "\n""\n", a2
                    print x-y
                    if -10 < x-y < 10:
                        aruco_points = a2
                        cv2.drawContours(img1, c1, i, (0,255,0), 2)
                        cv2.imshow('img',img1)
                        cv2.waitKey(500)
                        cv2.destroyAllWindows()
                        return aruco_points, '1'
        i = i + 1



    if aruco_points == []:
        return '-1', '-1'
```

```
def Perspective(aruco_points,img_name):
    """
    * Function Name:    Perspective
    * Input:        A numpy array with four points and name of an image
    * Output:       Returns the image after performing perspective transform
                        on it
    * Logic:        If the image points have a varied difference between the points, it mean
                        tilted and thus the points are mapped to proper set of points and ma
                        image plane.
    * Example Call: Perspective(points, "Marker.jpg")
    """
    img = cv2.imread(img_name)
    a1 = aruco_points
    print aruco_points
    if a1[0,0,0]>a1[1,0,0]:
        pts1 = np.float32([a1[1,0], a1[2,0], a1[3,0], a1[0,0]])
        pts2 = np.float32([[0,0], [0,300], [300,300], [300,0]])

    elif a1[0,0,0] < a1[1,0,0] :
        pts1 = np.float32([a1[0,0], a1[1,0], a1[2,0], a1[3,0]])
        pts2 = np.float32([[0,0], [0,300], [300,300], [300,0]])
    print pts1
    print pts2
    M = cv2.getPerspectiveTransform(pts1,pts2)

    dst = cv2.warpPerspective(img,M,(300,300))
    per_img = dst.copy()
    cv2.imshow("Perspective",dst)
    cv2.waitKey(500)
    cv2.destroyAllWindows()
    resize = cv2.resize(per_img, (399,399), interpolation = cv2.INTER_CUBIC)
    #Grid box
    dx, dy = 57,57

    # Custom (rgb) grid color
    grid_color = [0,0,255]

    # Modify the image to include the grid
    resize[:,::dy,:] = grid_color
    resize[::dx,:,:] = grid_color
    aruco = resize[57:342, 57:342]
    cv2.imshow("grid",aruco)
    cv2.waitKey(500)
```

```
    cv2.destroyAllWindows()
    ret, thresh= cv2.threshold(aruco, 127,255, cv2.THRESH_BINARY)
    g2 = cv2.cvtColor(thresh, cv2.COLOR_BGR2GRAY)
    return g2, pts1



def Crop(aruco_points,img_name):
"""
    * Function Name:    Crop
    * Input:        A numpy array with four points and name of an image
    * Output:       Returns the image after performing a refined perspective transform.
                        on it
    * Logic:        If the image points have a varied difference between the points, it mean
                        tilted and thus the points are mapped to defined set of points and n
                        image plane.
    * Example Call: Crop(points, "Marker.jpg")
    img = cv2.imread(img_name)
    a1 = aruco_points
    print aruco_points
    lx = -999
    ly = -999
    sx = 999
    sy = 999
    for i in range(0,4):
        if a1[i,0,0] > lx :
            lx = a1[i,0,0]
    print lx

    for i in range(0,4):
        if a1[i,0,0] < sx:
            sx = a1[i,0,0]
    print sx

    for i in range(0,4):
        if a1[i,0,1] > ly:
            ly = a1[i,0,1]
    print ly
    for i in range(0,4):
        if a1[i,0,1] < sy:
            sy = a1[i,0,1]
    print sy
    '''
    for i in range(0,4):
        if 0 < a1[i,0,0] - sx < 10:
```

```
            if a[i, 0, 1] == sy:
                a1[0,0] = [a1[i, 0, 0], sy]
                a1[1,0] = [sx,
    '''
    pts1 = np.float32([[sx,sy],[sx,ly],[lx,ly],[lx,sy]])
    pts2 = np.float32([[0,0],[0,300],[300,300],[300,0]])
    print pts1
    print pts2
    M = cv2.getPerspectiveTransform(pts1,pts2)

    dst = cv2.warpPerspective(img,M,(300,300))
    per_img = dst.copy()
    cv2.imshow("crop",dst)
    cv2.waitKey(500)
    cv2.destroyAllWindows()
    resize = cv2.resize(per_img, (399,399), interpolation = cv2.INTER_CUBIC)
    #Grid box
    dx, dy = 57,57

    # Custom (rgb) grid color
    grid_color = [0,0,255]

    # Modify the image to include the grid
    resize[:,::dy,:] = grid_color
    resize[::dx,:,:] = grid_color
    aruco = resize[57:342, 57:342]
    cv2.imshow("grid",aruco)
    cv2.waitKey(500)
    cv2.destroyAllWindows()
    ret, thresh= cv2.threshold(aruco, 127,255, cv2.THRESH_BINARY)
    g2 = cv2.cvtColor(thresh, cv2.COLOR_BGR2GRAY)
    return g2, pts1

def findArucoID(marker_img):
    """
    * Function Name:    findArucoID
    * Input:        An image of Aruco marker.
    * Output:       Returns an integer value that represents the ID of the
                        Aruco marker
    * Logic:        The second and fourth columns from the Aruco marker are
                        analyzed. If the pixel value in the grid cell is 0
                        (black), it is taken as 0. If it is 255(white), it is
                        considered as 1. The binary number is generated by
                        reading in a top-bottom, left-right manner.
    * Example Call: findArucoID(img)
    """
```

```
    height = 57
    width = 57
    ret_val = 0
    cv2.imshow("aruco", marker_img)
    cv2.waitKey(500)
    cv2.destroyAllWindows()
    for i in range(5):
        for y in (1, 3):
            px1 = marker_img[height*i + 29, y * width + width/2]
            if px1 == 255:
                val = 1
                #binary.append(val)
            else:
                val = 0
                #binary.append(val)
            ret_val = 2 * ret_val + val
    #print "Binary", binary, ret_val
    return ret_val


def sendPoints(x, y, t, t1, markerID):
    """
    * Function Name:    sendPoints
    * Input:        The (x,y) coordinates of a point, angle, t, which is the
                        rotation angle and the ID(integer) of the marker whose
                        points are being sent.
    * Output:       Sends a message to the server containing the information
                        from the input
    * Logic:        A TCP message is sent to the server through socket
                        programming.
    * Example Call: sendPoints(269, 346, 1.11812, 34)
    """
    global s


    msg = str(x) + " " + str(y) + " " + str(t) + " "+str(t1) + " "+ str(markerID)
    print "message sent"


    s.send(msg)



def getProperties(points):
    """
```

```
    * Function Name:    getProperties
    * Input:        A set of four points of the corners of aruco markers.
                        These points can be obtained from is_aruco_present()
                        function.
    * Output:       -
    * Logic:        The Perspective-n-Point problem is solved by Ransac
                        algorithm. We obtain the translation and rotation
                        vectors through this function.
    * Example Call: getProperties(points)
    """

    global objp

    # Arrays to store object points and image points from all the images.
    objpoints = objp
    print "OBJP", objpoints

    imgpoints = points

    #imgpoints = np.array(imgpoints)
    print "IMGP", imgpoints

    mtx = np.load('matrix.npy')
    dist = np.load('distortion.npy')

    rvec, tvec, inliers = cv2.solvePnPRansac(objpoints, imgpoints, mtx, dist)
    print "Rvec\n", rvec
    print "\nTvec", tvec

    x = tvec[0][0]
    y = tvec[2][0]

    dst, jacobian = cv2.Rodrigues(rvec)

    print "Rot Matrix", dst

    t = math.asin(-dst[0][2])
    t1 = math.acos(dst[0][0])

    return x, y, t, t1


def Video(True):
    """
    * Function Name:    Video
    * Input:        -
```

```
* Output:           Analyzes and detects markers till all markers are
                    detected.
* Logic:        It runs through a infinite loop checking
                    every frame for markers. If marker is obtained,
                    it increments a count, else, next frame is analyzed.
                    This process is continued till all the markers are
                    detected.
* Example Call: Video(True)
"""
global count,s, mark_detect
cap = cv2.VideoCapture(1)

while(True):
    ret, frame = cap.read()
    cv2.imshow("Video", frame)
    cv2.waitKey(500)
    aruco_points, flag = is_aruco_present(frame)
    #cv2.imshow("Captured", frame)
    if flag != '-1':
        img_name = "Marker.jpg"
        cv2.imwrite(img_name, frame)
        #count = count + 1
        if flag == '1':
            p_img, pts = Crop(aruco_points, img_name)
        elif flag == '2':
            p_img, pts = Perspective(aruco_points, img_name)
        m_id = findArucoID(p_img)
        if (m_id == 65 and mark_detect[0] == 0):
            mark_detect[0] = 1
        elif (m_id == 796 and mark_detect[1] == 0):
            mark_detect[1] = 1
        elif (m_id == 500 and mark_detect[2] == 0):
            mark_detect[2] = 1
        elif (m_id == 250 and mark_detect[3] == 0):
            mark_detect[3] = 1
        else:
            continue
        count = count + 1
        x, y, t, t1 = getProperties(pts)
        if (x == 0.0 and y == 0.0):
            count = count - 1
            if (m_id == 65 and mark_detect[0] == 1):
                mark_detect[0] = 0
            elif (m_id == 796 and mark_detect[1] == 1):
                mark_detect[1] = 0
            elif (m_id == 500 and mark_detect[2] == 1):
```

```
                    mark_detect[2] = 0
                elif (m_id == 250 and mark_detect[3] == 1):
                    mark_detect[3] = 0
                continue

            sendPoints(x, y, t, t1, m_id)
            print "X", x, "Y", y, "T", t, "T1", t1,  "ID", m_id
            if count == MAX_MARKERS:
                print "All detected"
                print "Closing connection"
                if cv2.waitKey(1) == 32: # Ascii for spacebar
                    s.send('q')
                break




    cap.release()
    cv2.waitKey(0)     # Escape sequence
    cv2.destroyAllWindows()

Video(True)
s.close()
```

The server will have the following code-

```
"""
This code receives and maps points

Authors: Niharika Jayanthi, Dheeraj Kamath
Project: Marker-based Localization
Mentor: Sanam Shakya

Main functions: draw_arena(), draw_marker(), draw_robot()
                getCoordinates(), get_socket()

Global variables: arena_length, arena_breadth, s, host, port, room_width

"""
import socket
import cv2
from matplotlib import pyplot as plt
import numpy as np
import math
```

```
#Define Globals

arena_length=600
arena_breadth=600

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostname()
port = 7000
s.bind((host, port))
s.listen(2)

room_width = 1160


#Helper functions

def draw_arena(img1):
    """
    * Function Name:    draw_arena
    * Input:        Image to be drawn in
    * Output:       Modified image with arena drawn
    * Logic:        The arena lines are drawn one at a time, with a distance
                        of 50 units separating them.
    * Example Call: draw_arena(image)
    """
    # Draw a diagonal blue line with thickness of 5 px
    row_width = 50
    col_width =  50

    #print arena_length/50
    for i in range(0, arena_length/row_width):
        cv2.line(img1,(row_width,0),(row_width,arena_length),(0,0,0),1)
        cv2.line(img1,(0,col_width),(arena_breadth,col_width),(0,0,0),1)
        row_width=row_width+50
        col_width=col_width+50


def draw_marker(id, img1):
    """
    * Function Name:    draw_marker
    * Input:        Marker ID, image to be drawn in
    * Output:       Returns 1 if valid marker is drawn. Else, it returns -1.
    * Logic:        The marker id is checked for validity. If marker is
                        valid, it draws in fixed position and returns 1. Else,
                        it returns -1.
```

```
    * Example Call: draw_marker(65, image)
    """
    marker_width = 50
    marker_length = 50

    font = cv2.FONT_HERSHEY_SIMPLEX

    if id == 65:
        x = 0
        y = 0
    elif id == 250:
        x = 550
        y = 0
    elif id == 796:
        x = 0
        y = 550
    elif id == 500:
        x = 550
        y = 550
    else:
        return '-1'

    cv2.rectangle(img1,(x,y),(x+marker_width,y+marker_length),(255,0,0,10),-1)
    cv2.putText(img1,'#'+str(id),(x+10,y+30), font, 0.5,(0,0,0),1)

    return 1


def draw_robot(x, y, theta_radian, img1):
    """
    * Function Name:    draw_robot()
    * Input:        x,y coordinates of the robot's position in map, angle
                        of inclination(theta) and image to be drawn in.
    * Output:       Modified image with the robot drawn in it and result is
                        displayed.
    * Logic:        The end point of the line is found by calculating a
                        rotation matrix and translating it. The robot is then
                        drawn as a circle and the line on the robot depicts
                        its orientation.
    * Example Call: draw_robot(250, 250, 45, image)
    """
    radius = 20

    rotation_matrix= [[np.cos(theta_radian), -np.sin(theta_radian)],
                      [np.sin(theta_radian), np.cos(theta_radian)]]
    R = np.array(rotation_matrix)
```

```
    xy = [[radius],[0]]
    xy = np.array(xy)
    rotated_xy = np.dot(R,xy)

    translation = [[x],[y]]
    translation = np.array(translation)
    trans_xy = rotated_xy + translation

    #Convert from floating point to integers
    x = int(x)
    y = int(y)

    cv2.circle(img1,(x,y), radius, (0,0,255), -1)

    cv2.line(img1,(x,y),(trans_xy[0],trans_xy[1]),(0,0,0),2)
    cv2.imshow("Position", img1)
    cv2.waitKey(1000)


def getCoordinates(x, y, t, mID):
    """
    * Function Name:    getCoordinates
    * Input:        x, y coordinates of point, angle t, marker ID
    * Output:       Returns new values of x, y and t.
    * Logic:        It compares the marker ID to a valid list of markers,
                        whose position in a room is already known to us and
                        returns the values of x,y and t according to the ID.
    * Example Call: getCoordinates(125, 235, 45, 500)
    """

    if mID == 65:
        return x, y, 3*math.pi/2 - abs(t)
    elif mID == 796:
        return x, 550 - y, math.pi/2 + abs(t)
    elif mID == 500:
        return 550 - x, 550 - y, math.pi/2 - abs(t)
    elif mID == 250:
        return 550 - x, y, 3 * math.pi/2 + abs(t)
    else:
        print "Marker doen't match!"
        return 0, 0, 0



def get_socket():
    """
```

```
* Function Name:    get_socket
* Input:        -
* Output:       -
* Logic:        This function creates a TCP socket and receives
                    information from the client. This information includes
                    x, y coordinates of a marker, the angle t(calculated
                    with sine inverse), angle t1 (calculated as cosine
                    inverse) and the marker ID detected. These values are
                    passed to getCoordinates, which returns the x,y
                    values scaled to virtual map. Then, the marker/arena and
                    robot is drawn.
* Example Call: get_socket()
"""
global s, first_msg, arena_M, cur_x, cur_y

c, addr = s.accept()

while True:
    msg = c.recv(100)
    print "Message received is", msg

    try:
        if msg == 'q':
            print "End of messages.\n"
            break

        x, y, t, t1, m_id = msg.split()
        x = float(x)
        y = float(y)
        t = float(t)
        t1 = float(t1)
        m_id = int(m_id)

        x = 600 * (x/room_width)
        y = 600 * (y/room_width)

        Rx = abs(y * math.cos(math.pi/2 - t))
        Ry = abs(y * math.sin(math.pi/2 - t))
        img_arena = arena_M.copy()
        ret = draw_marker(m_id, arena_M)
        if ret == '-1':
            print "Marker not recognised."

        print "X", x, "Y", y, "T", t, "T1", t1, "mID", m_id

        print "Rx", Rx, "Ry", Ry
```

```
            mx, my, t = getCoordinates(Rx, Ry, t, m_id)

            if (mx,my,t) == (0,0,0):
                print "Invalid coordinates"
                continue
            arena_copy = arena_M.copy()
            draw_robot(mx, my, t, arena_copy)

        except ValueError:
            print "Bad message!\n"
            break

# Create a black image
img = np.ones((arena_length,arena_breadth,4), np.uint8)*245
#img2 = np.ones((arena_length,arena_breadth,4), np.uint8)*255

draw_arena(img)
arena_M = img
get_socket()
s.close()

#cv2.imshow("Map", arena_M)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Resources

- Server code - Github link
- Client code - Github link