

Universidad Politécnica de Atlacomulco
Ingeniería en Robótica

**PROYECTO DE ESTADÍA
“APLICACIÓN DE ROBOTS MÓVILES EN UNA
FORMACIÓN DE ROBOTS MÓVILES
DIFERENCIALES”**

Presentado por:
Francisco Javier Meléndez Ruiz

Profesor Asesor:
Dr. Erick Axel Padilla García

Introducción

Introducción

Nuestros primeros esfuerzos son puramente instintivos, de una imaginación vivida e indisciplinada

Introducción

Nuestros primeros esfuerzos son
puramente instintivos, de una
imaginación vivida e indisciplinada

- Nikola Tesla

Objetivo

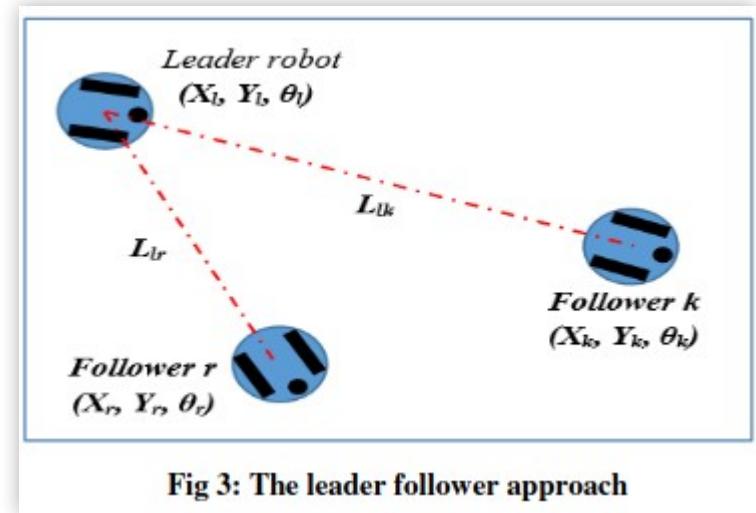
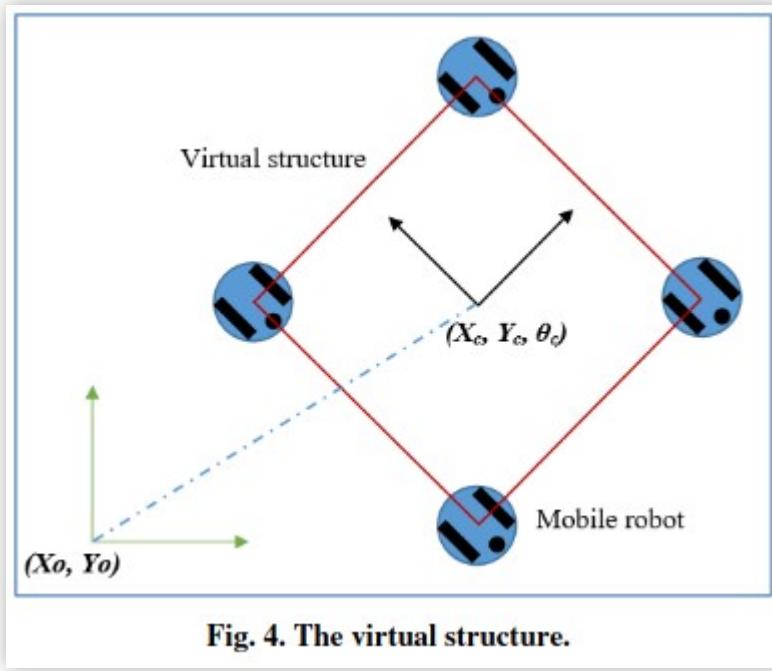
- Implementar un método de bajo costo para la formación de robots móviles diferenciales usando una cámara.

Marco Teórico

- Formaciones

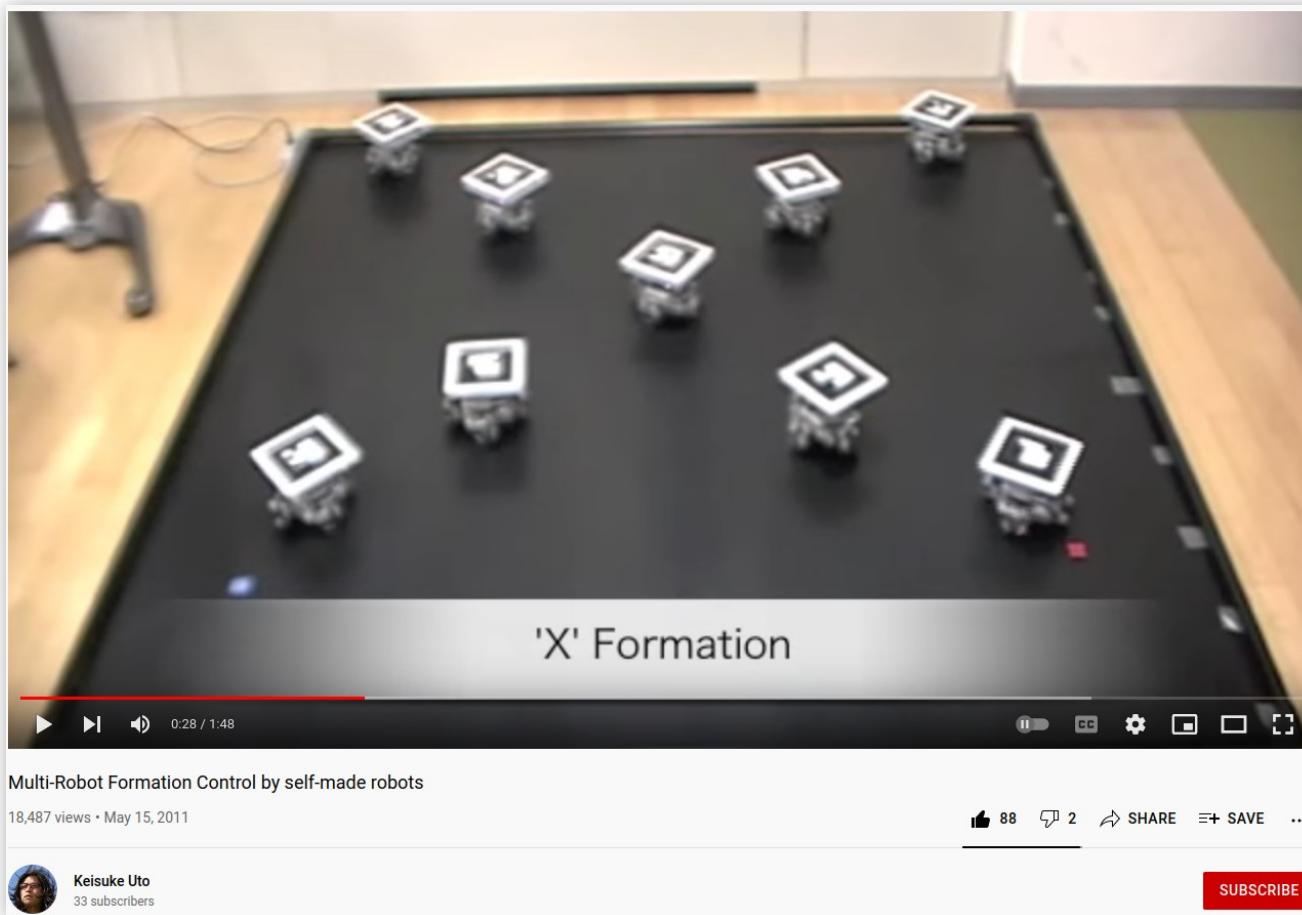
Marco Teórico

- Formaciones



Marco Teórico

- Formaciones



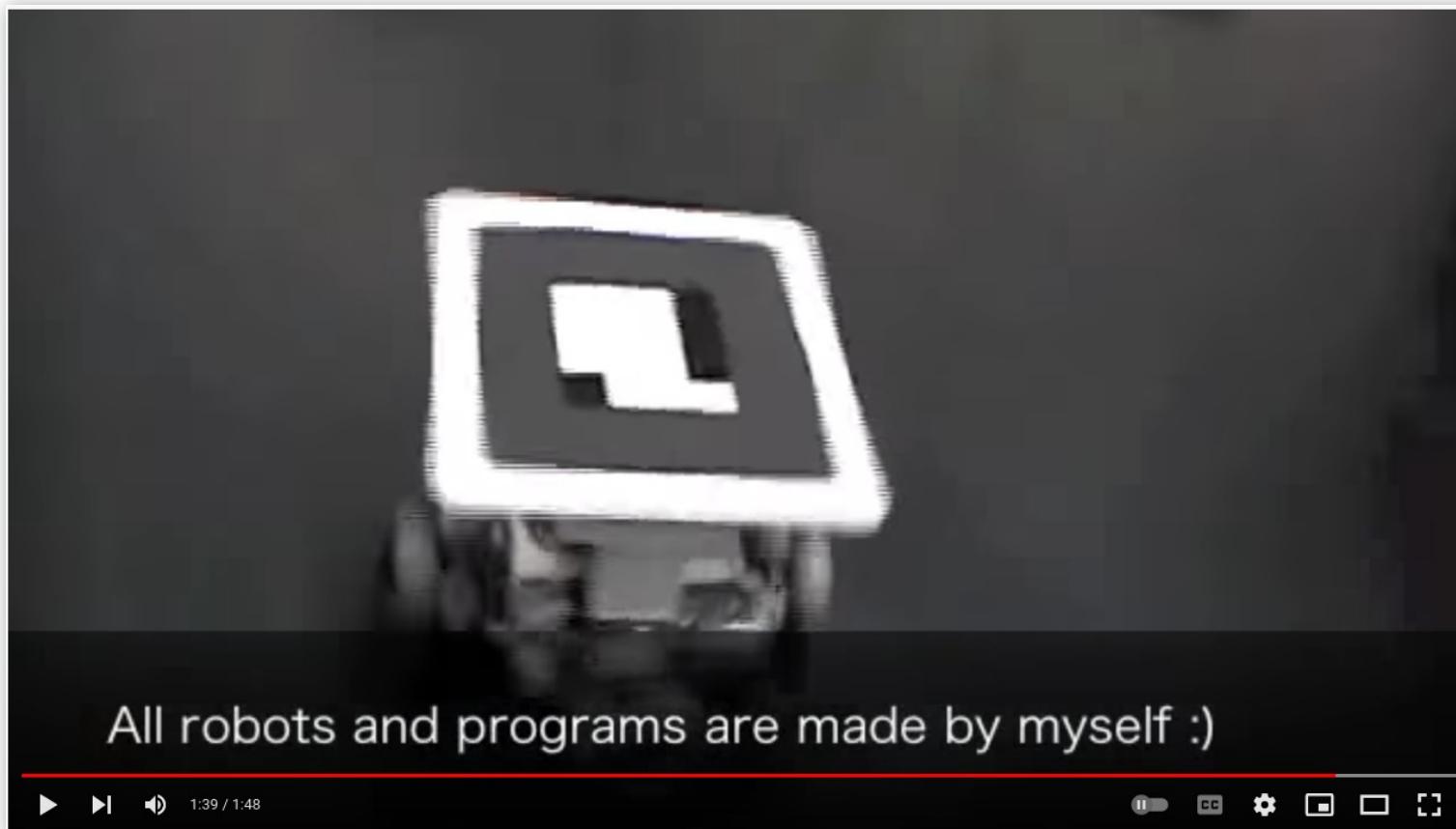
Marco Teórico

- Formaciones



Marco Teórico

- Formaciones



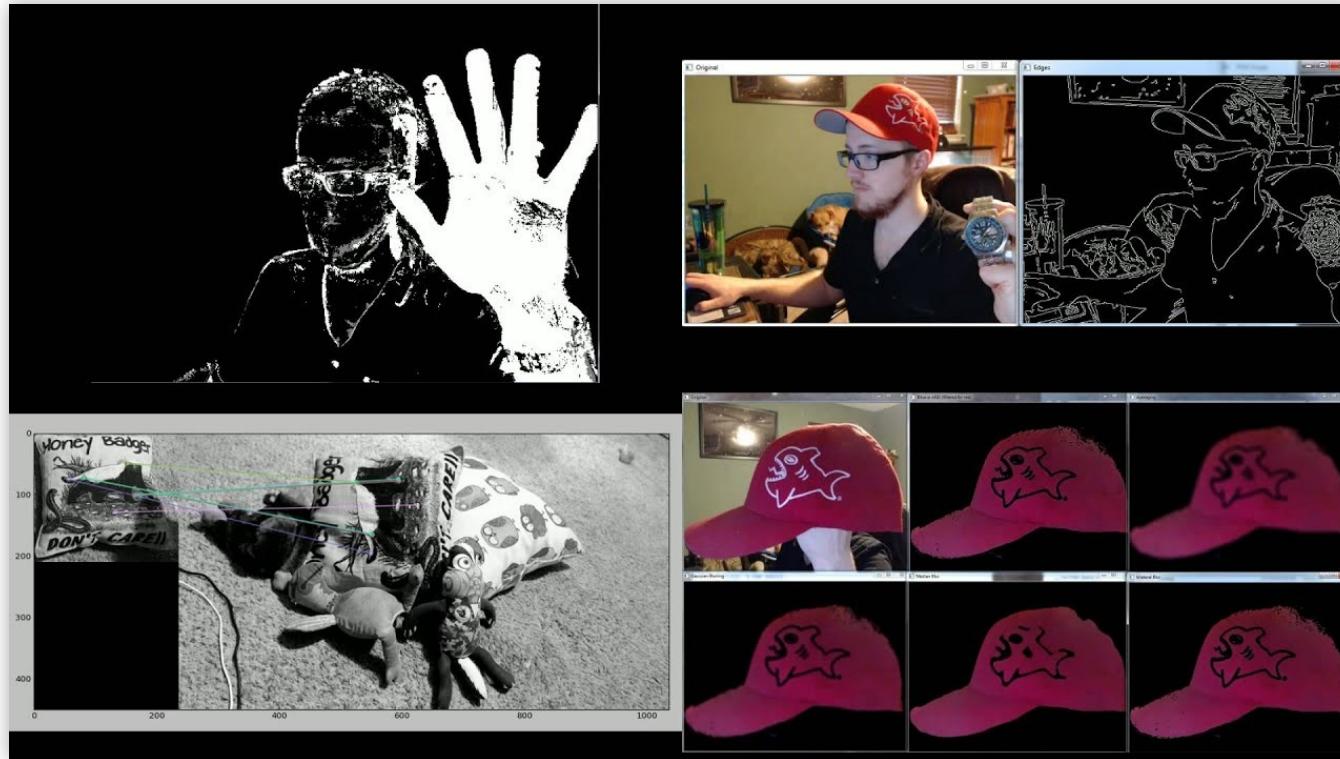
Marco Teórico

- OpenCV y visión



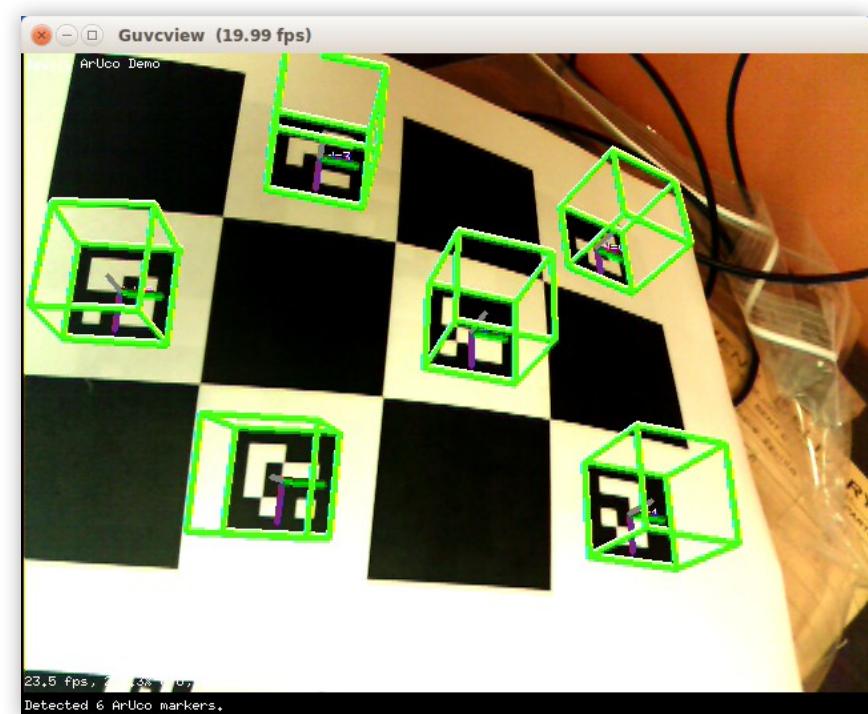
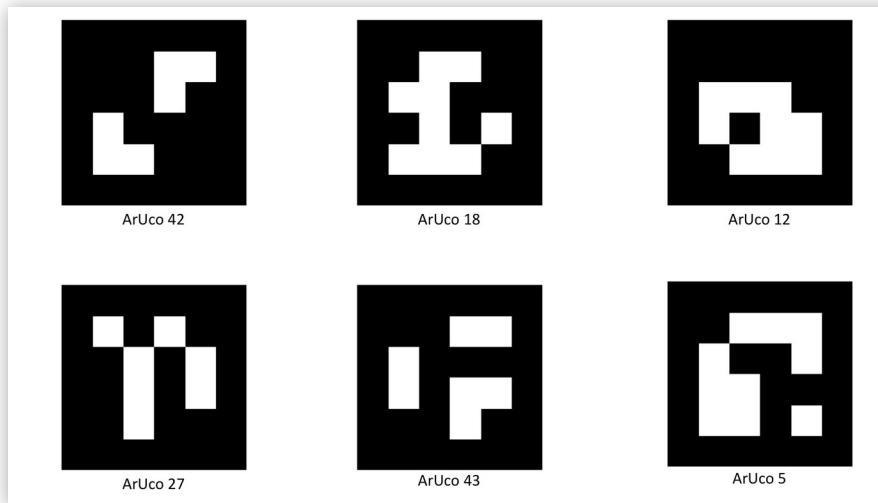
Marco Teórico

- OpenCV y visión



Marco Teórico

- OpenCV y visión



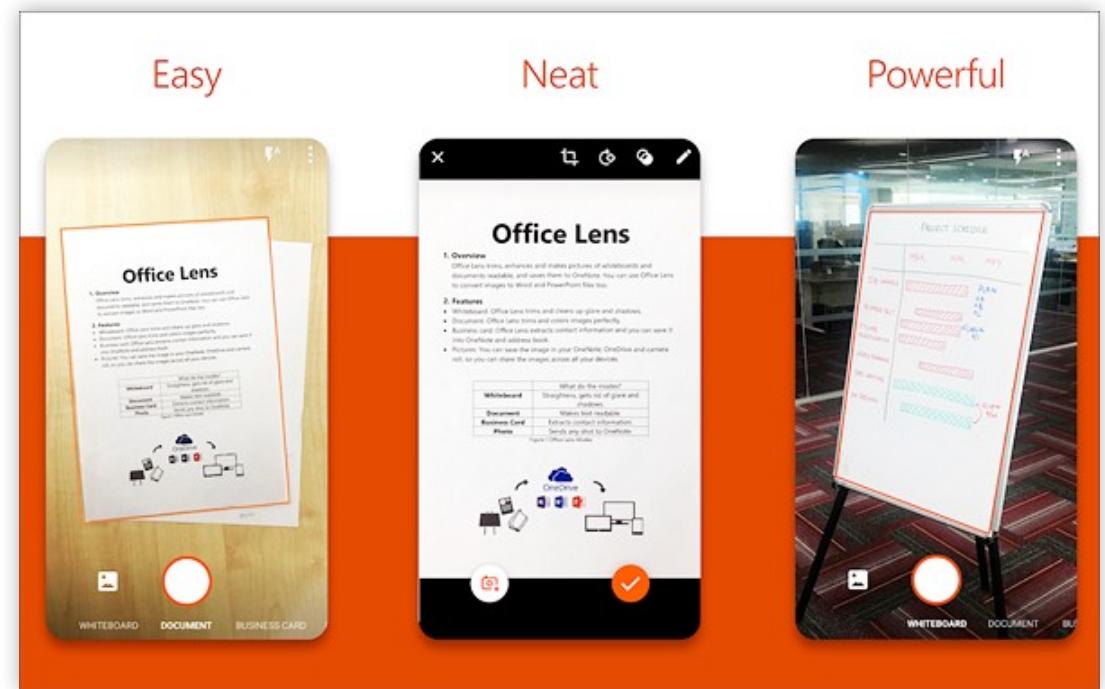
Marco Teórico

- OpenCV y visión



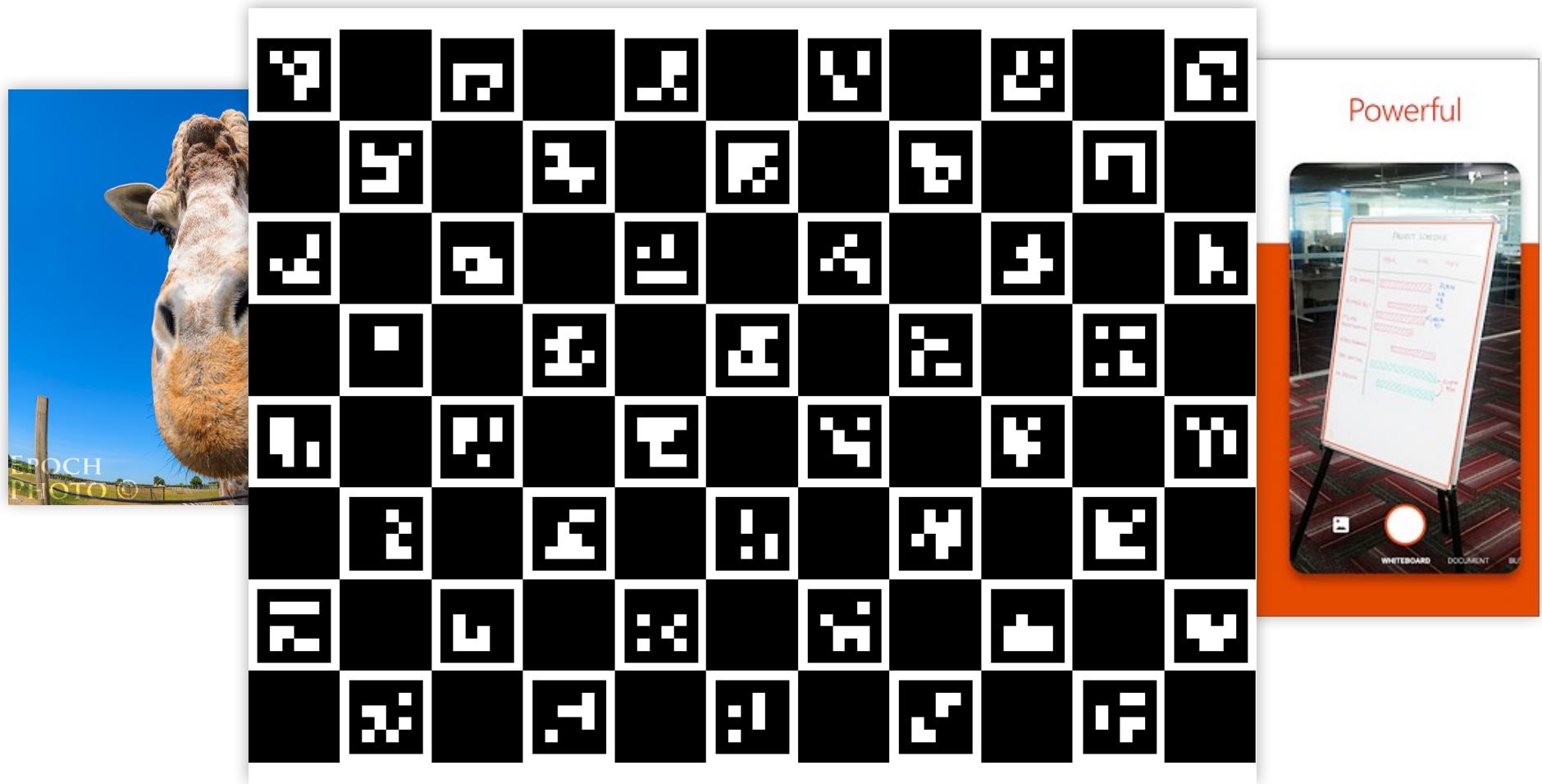
Marco Teórico

- OpenCV y visión



Marco Teórico

- OpenCV y visión

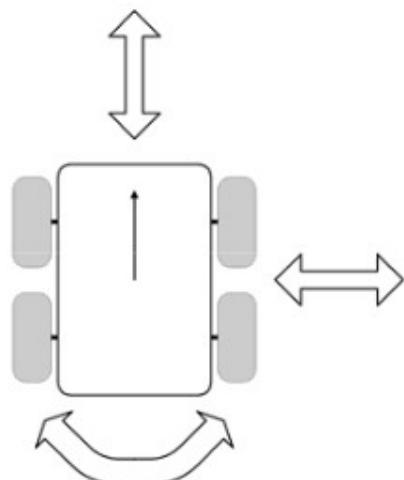


Marco Teórico

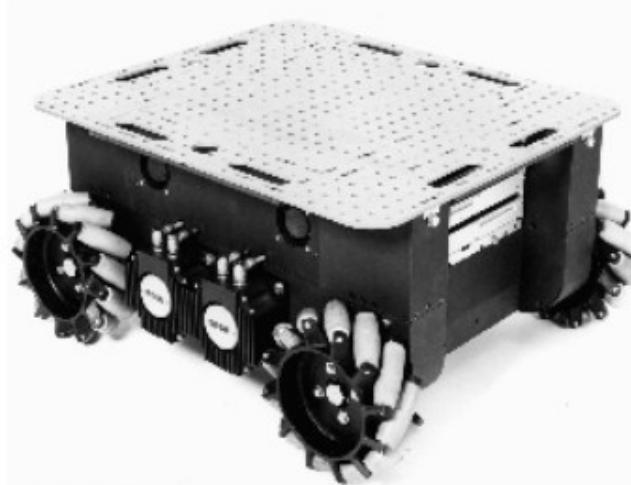
- Robots móviles

Marco Teórico

- Robots móviles



(a)



(b)

Figura 2.4.1. Robot omnidireccional con ruedas suecas. (a) Maniobrabilidad. (b) Robot Uranus (Universidad de Michigan).

Marco Teórico

- Robots móviles

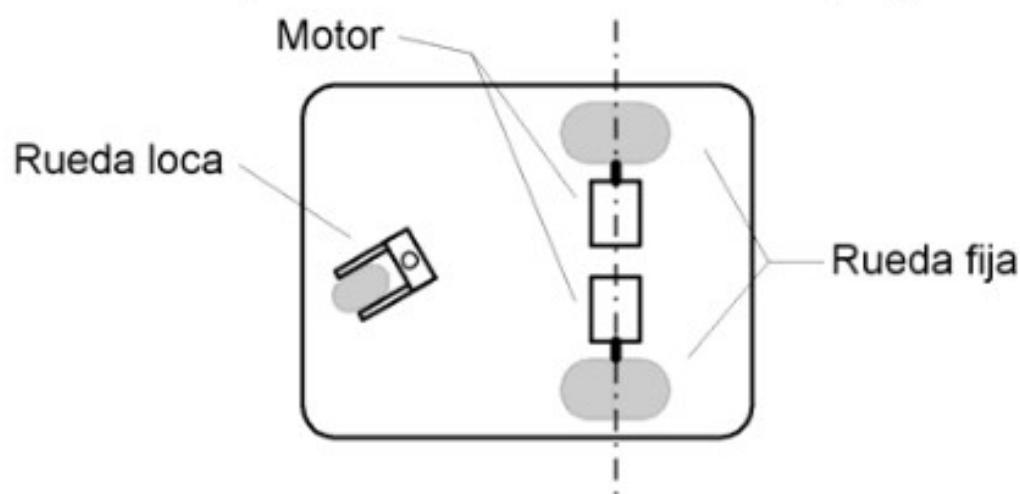


Figura 2.4.4. Uniciclo. (a) Estructura. (b) Robot Pioneer (ActiveMedia).

Marco Teórico

- Robots móviles

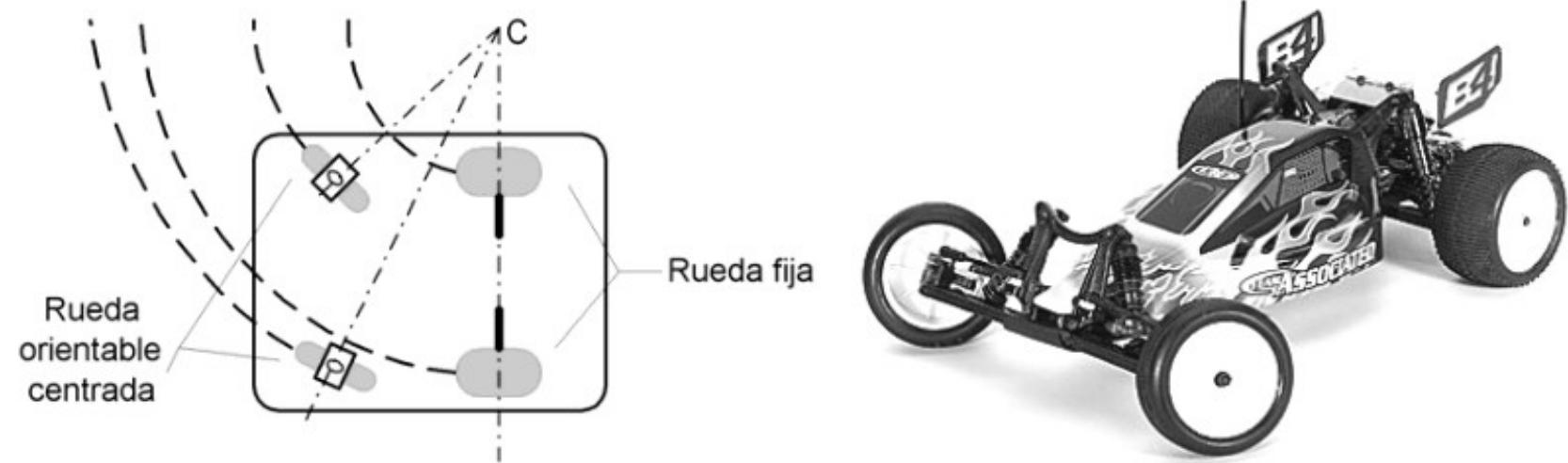


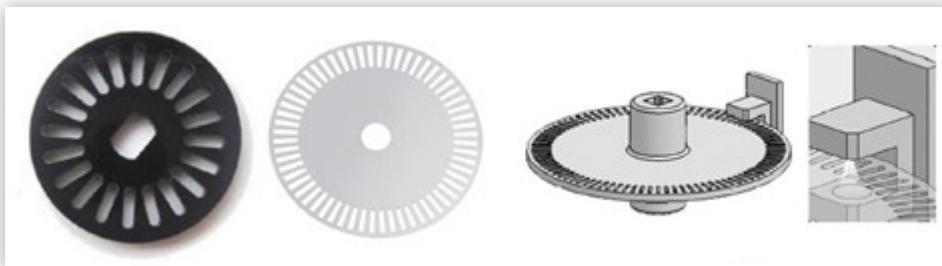
Figura 2.4.6. Sistema de dirección Ackerman.

Marco Teórico

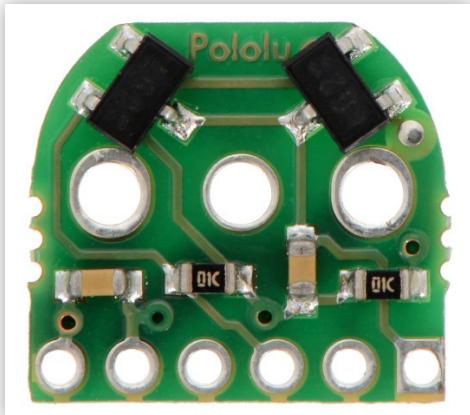
- Robots móviles
 - Encoders

Marco Teórico

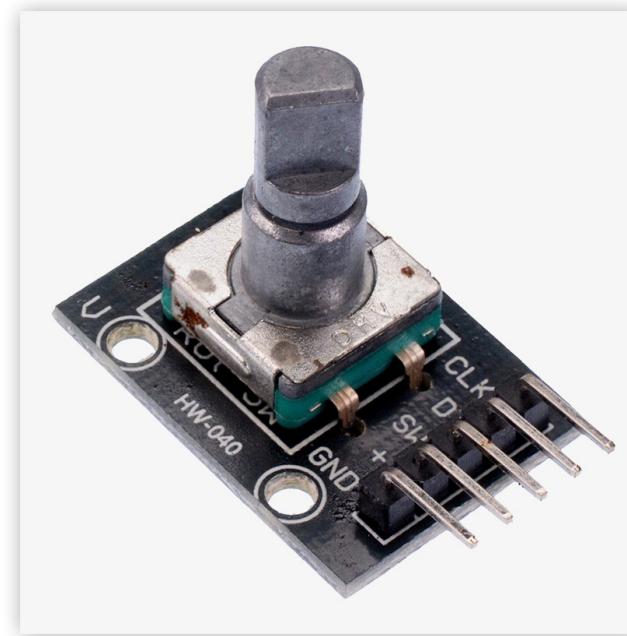
- Encoders



Encoder óptico



Encoder con sensores
de efecto hall



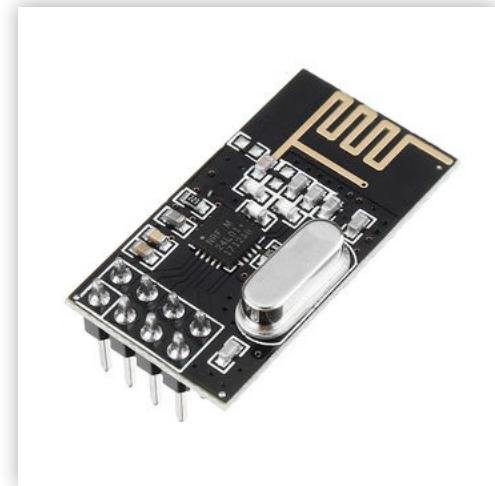
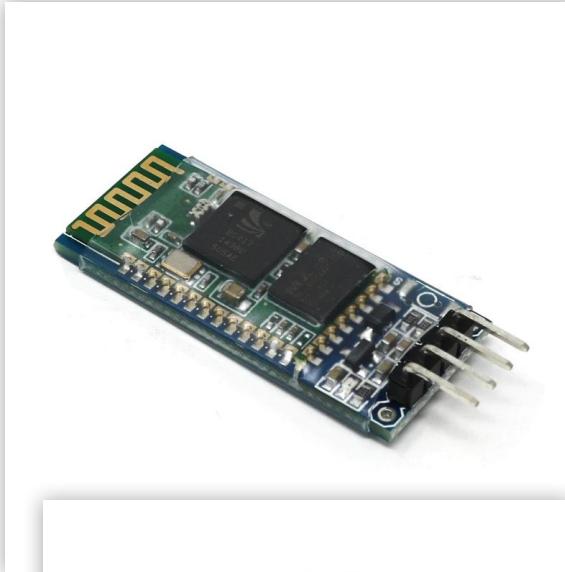
Encoder rotatorio

Marco Teórico

- Robots móviles
 - Comunicaciones

Marco Teórico

Comunicaciones



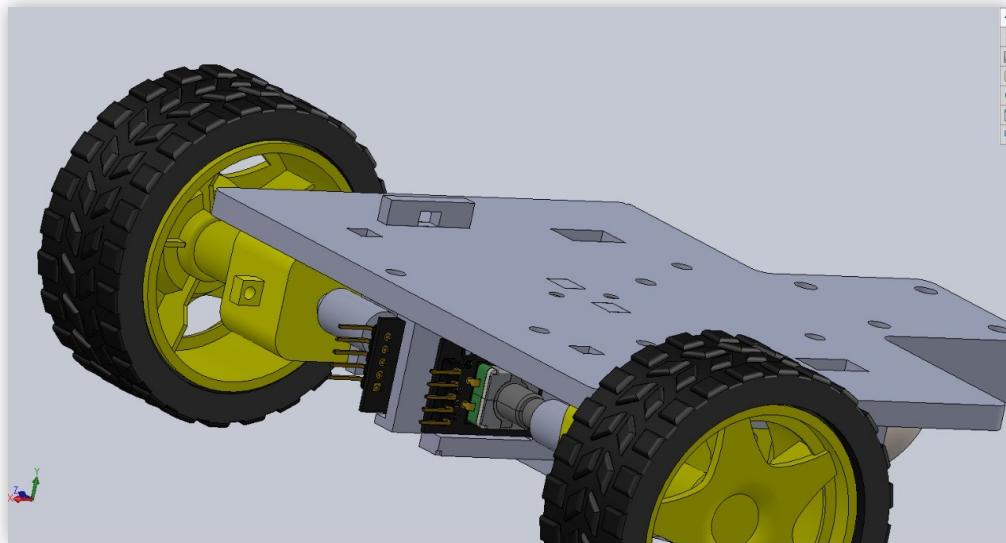
Procedimientos y pruebas

Procedimientos y pruebas

- Materiales

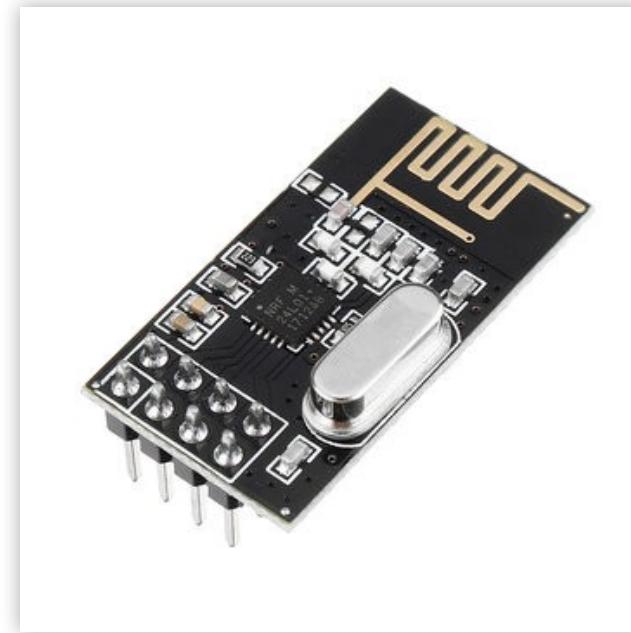
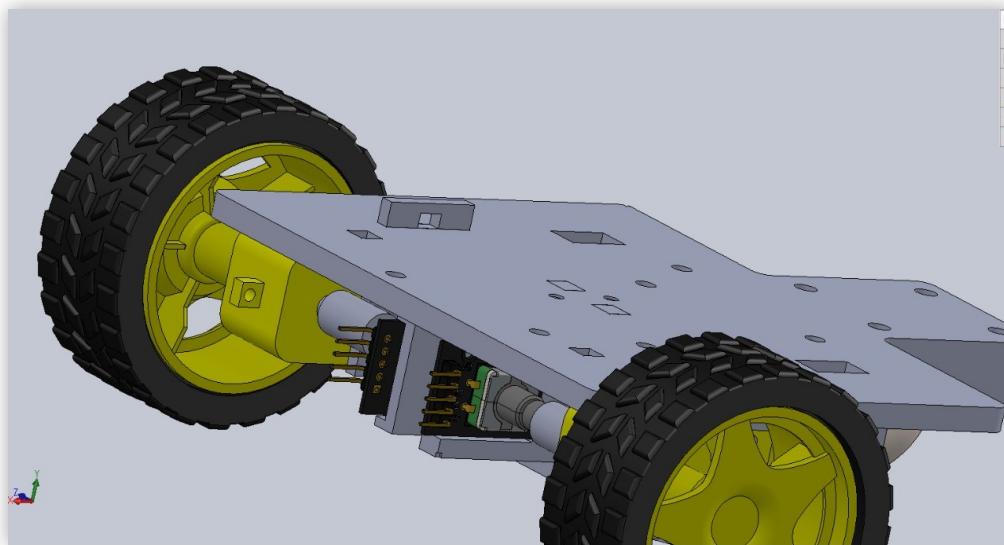
Procedimientos y pruebas

- Materiales



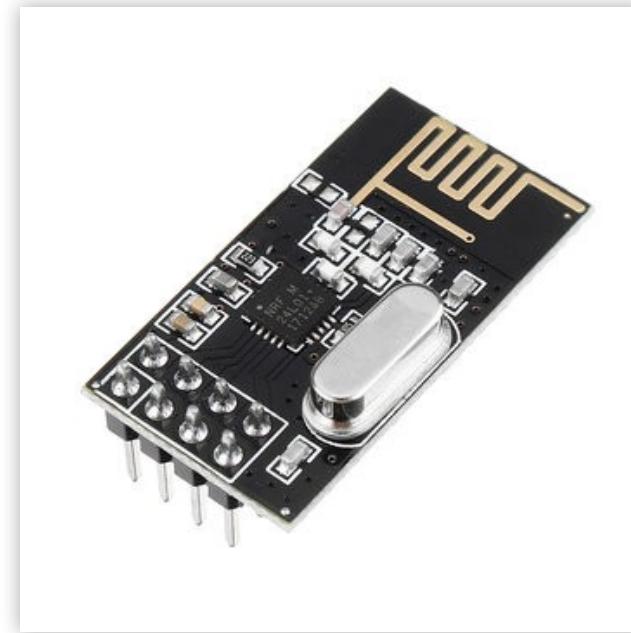
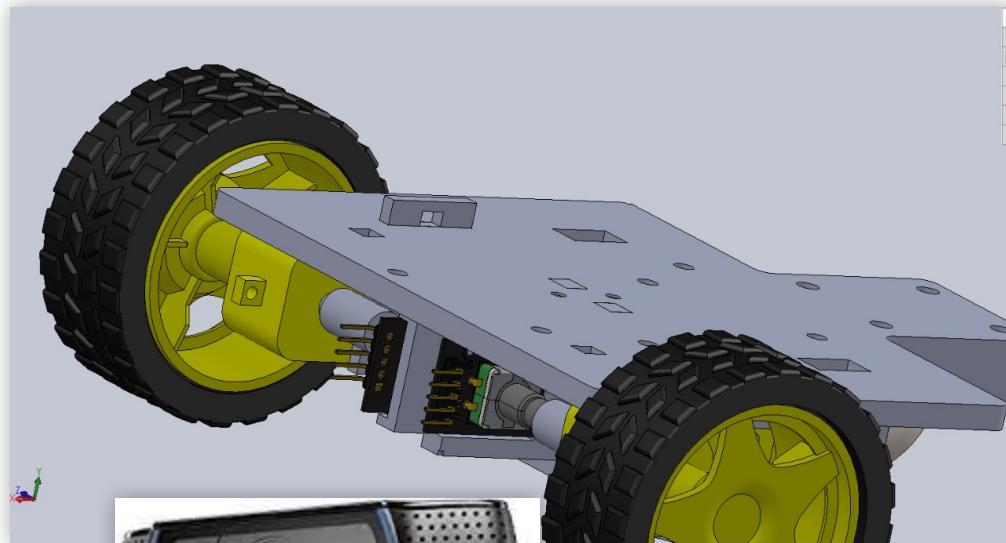
Procedimientos y pruebas

- Materiales



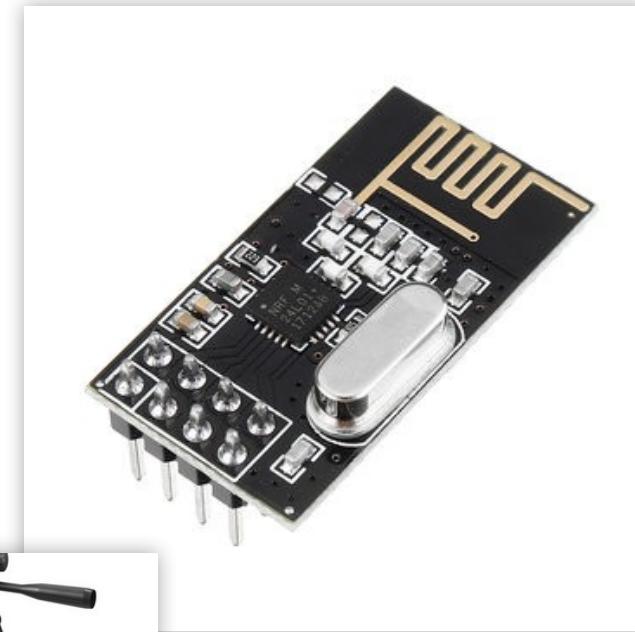
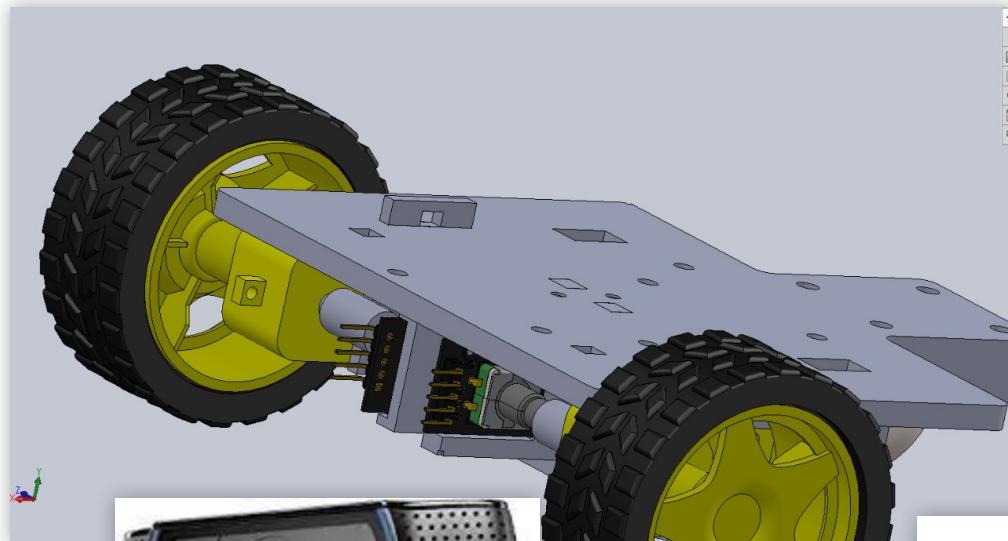
Procedimientos y pruebas

- Materiales



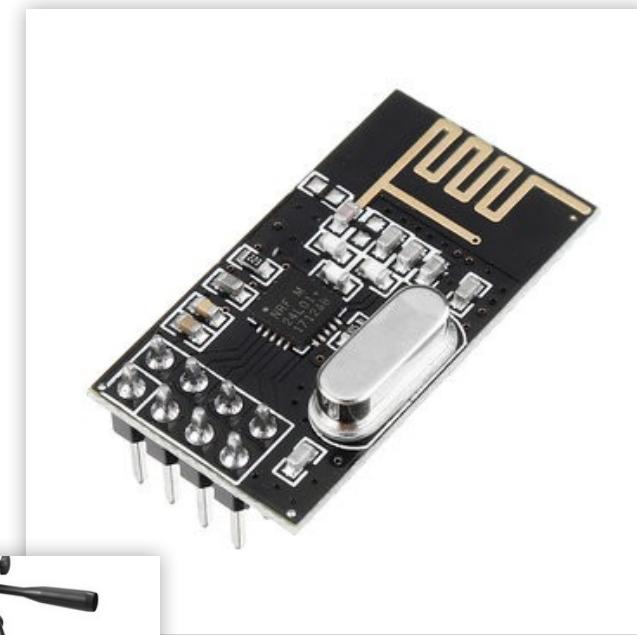
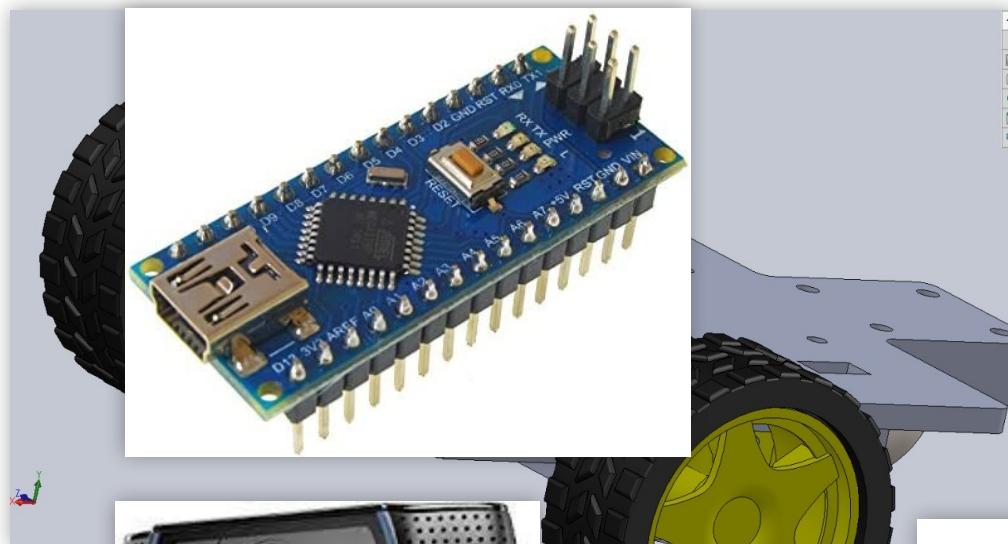
Procedimientos y pruebas

- Materiales



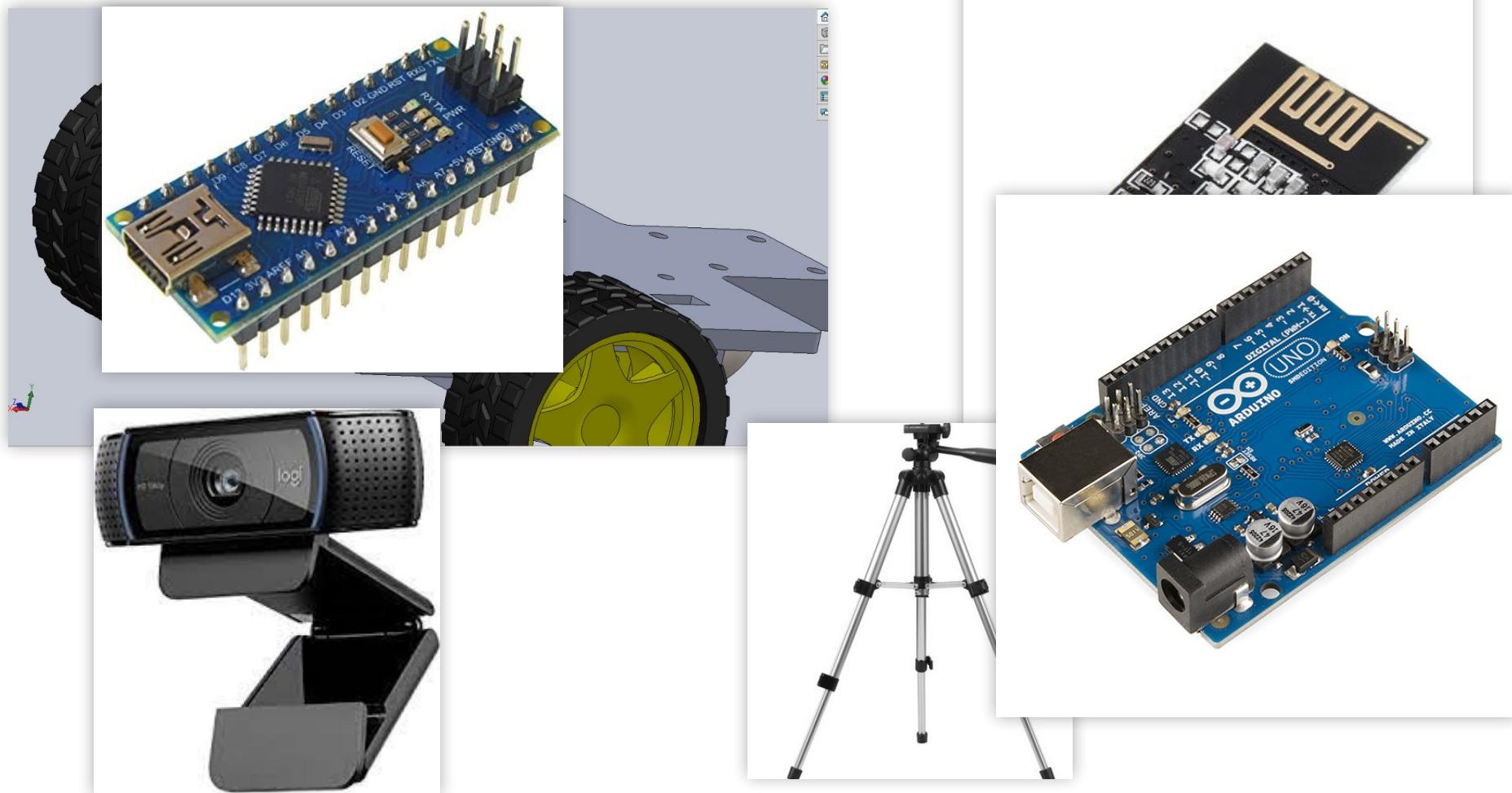
Procedimientos y pruebas

- Materiales



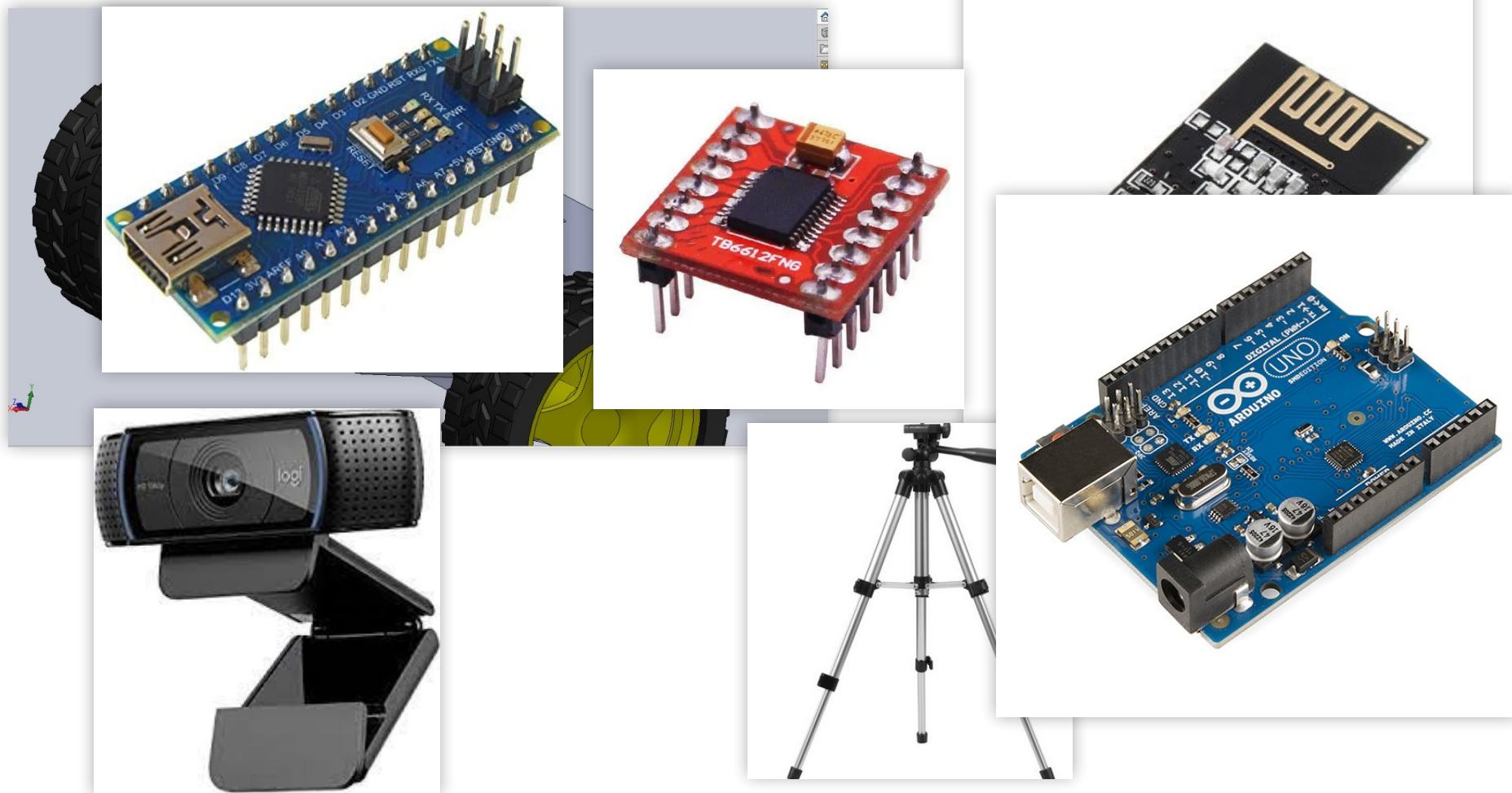
Procedimientos y pruebas

- Materiales



Procedimientos y pruebas

- Materiales



Procedimientos y pruebas

- Materiales

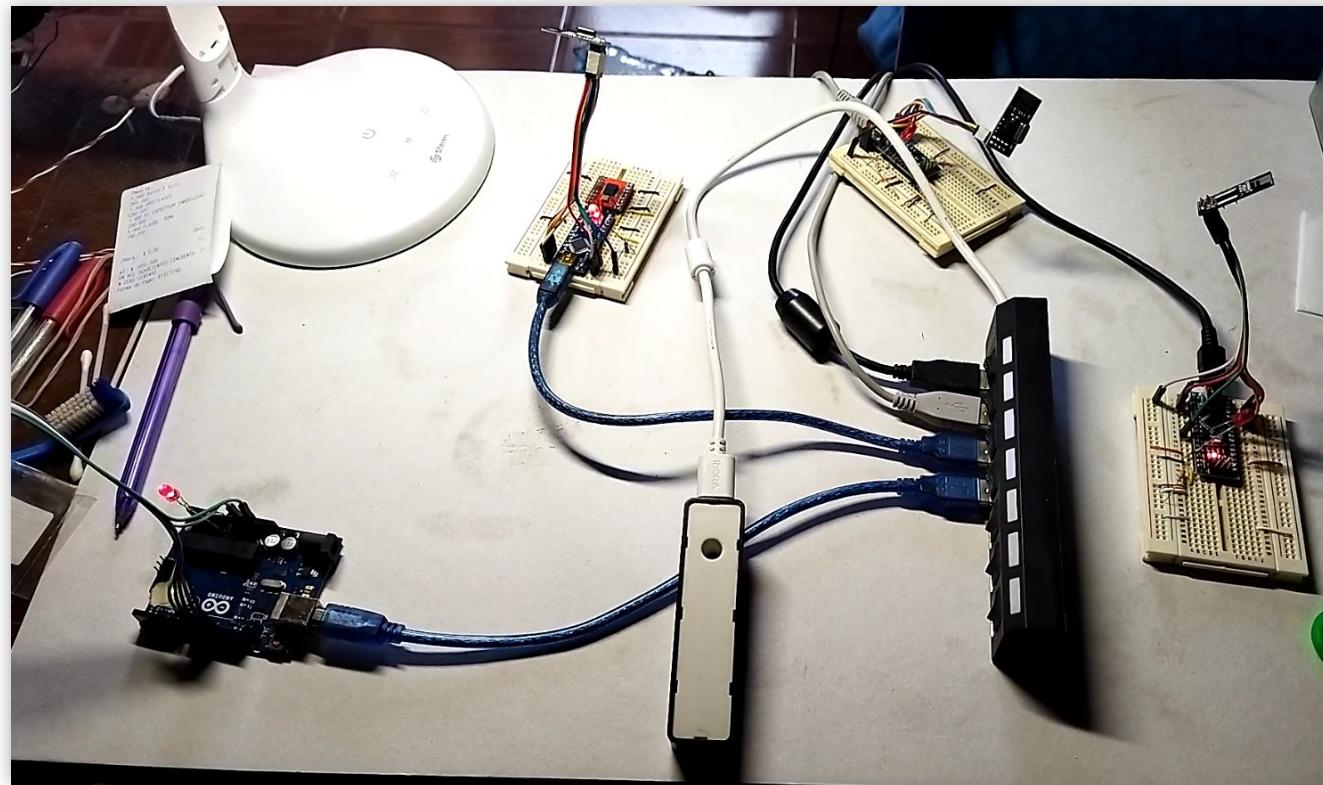


Procedimientos y pruebas

- Pruebas de comunicación entre Arduinos con el módulo NRF24L01

Procedimientos y pruebas

- Pruebas de comunicación entre Arduinos con el módulo NRF24L01



Procedimientos y pruebas

- Impresión de los marcadores ArUco

Procedimientos y pruebas

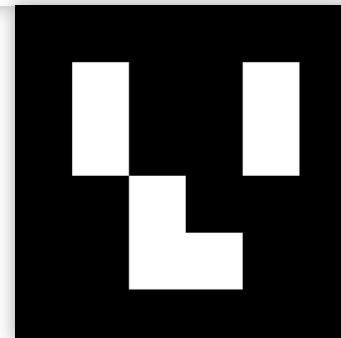
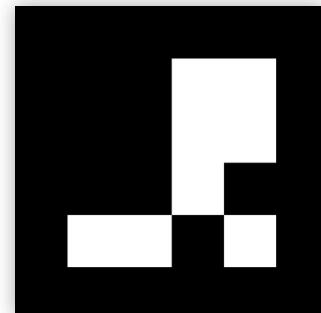
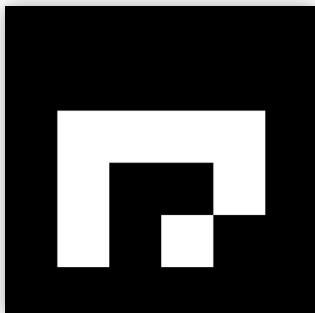
- Impresión de los marcadores ArUco

```
arucoGenerate.py > ...
1 import numpy as np
2 import cv2
3
4 dictionary = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
5 markerImage1 = cv2.aruco.drawMarker(dictionary, 1, 200)
6 markerImage2 = cv2.aruco.drawMarker(dictionary, 2, 200)
7 markerImage3 = cv2.aruco.drawMarker(dictionary, 3, 200)
8
9 # cv2.imshow("markerImage", markerImage1)
10 cv2.imwrite("aruco1.png", markerImage1)
11 cv2.imwrite("aruco2.png", markerImage2)
12 cv2.imwrite("aruco3.png", markerImage3)
13 # cv2.waitKey(0)
14
```

Procedimientos y pruebas

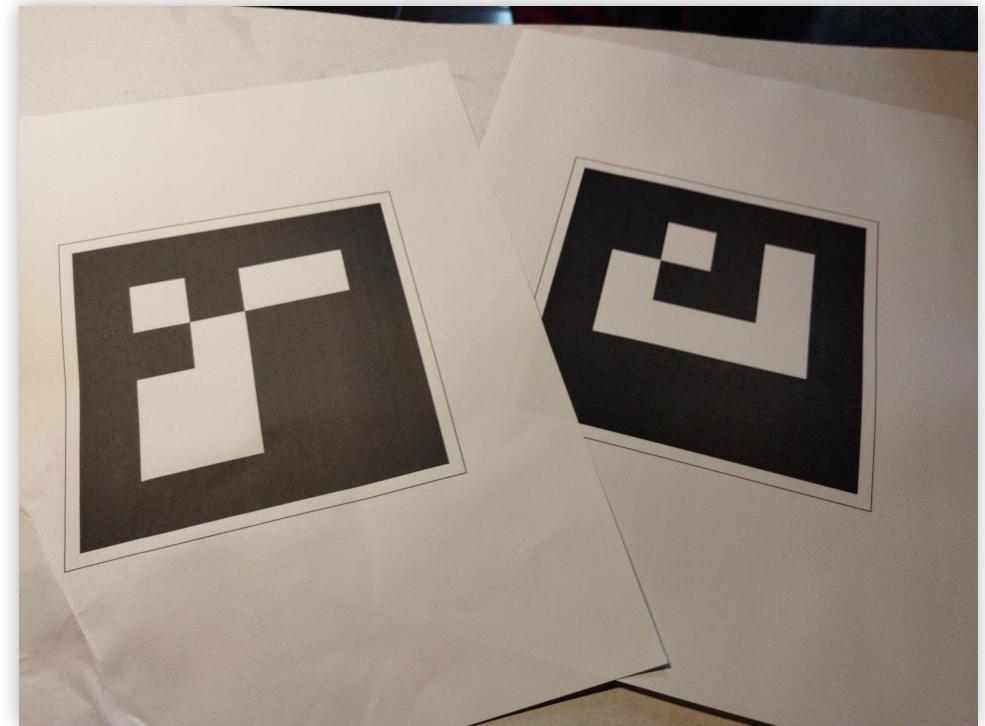
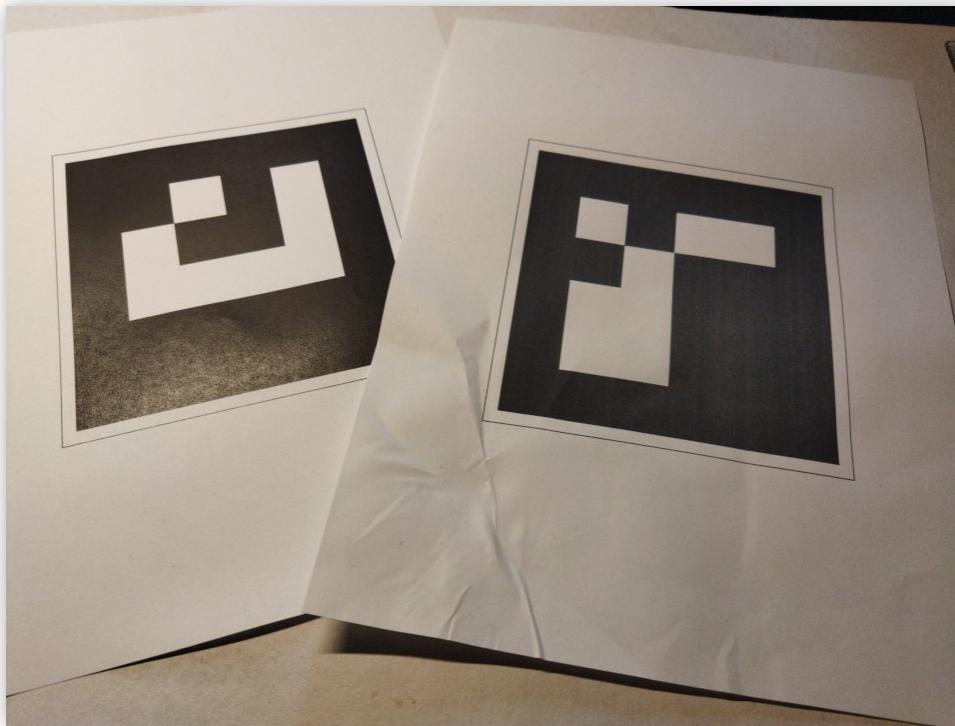
- Impresión de los marcadores ArUco

```
arucoGenerate.py > ...
1 import numpy as np
2 import cv2
3
4 dictionary = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
5 markerImage1 = cv2.aruco.drawMarker(dictionary, 1, 200)
6 markerImage2 = cv2.aruco.drawMarker(dictionary, 2, 200)
7 markerImage3 = cv2.aruco.drawMarker(dictionary, 3, 200)
8
9 # cv2.imshow("markerImage", markerImage1)
10 cv2.imwrite("aruco1.png", markerImage1)
11 cv2.imwrite("aruco2.png", markerImage2)
12 cv2.imwrite("aruco3.png", markerImage3)
13 # cv2.waitKey(0)
14
```



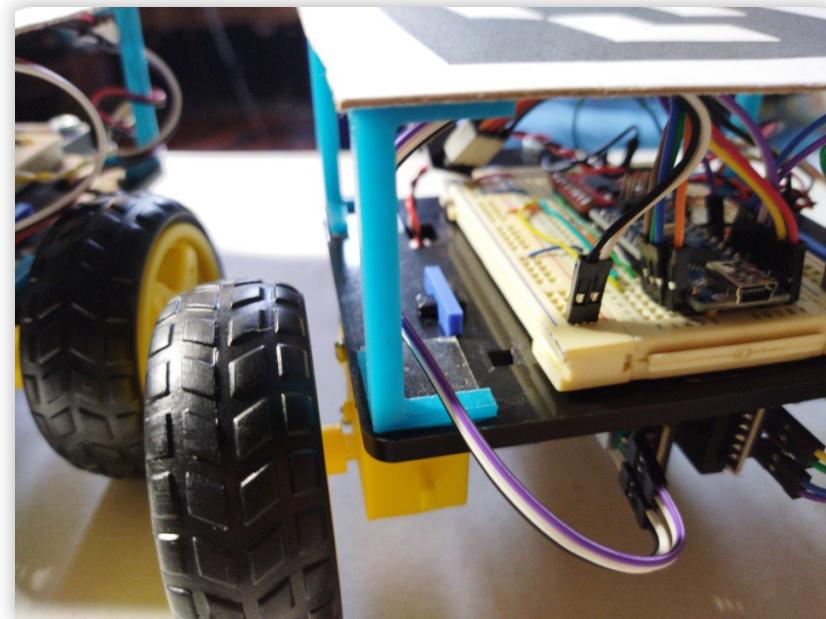
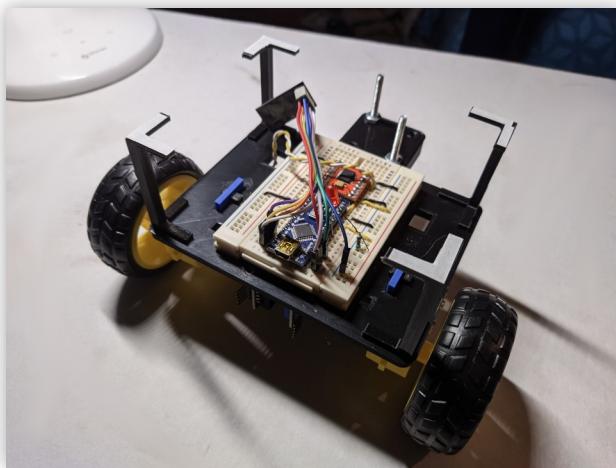
Procedimientos y pruebas

- Impresión de los marcadores ArUco



Procedimientos y pruebas

- Impresión de los marcadores ArUco



Procedimientos y pruebas

- Detección de ArUco con OpenCV

Procedimientos y pruebas

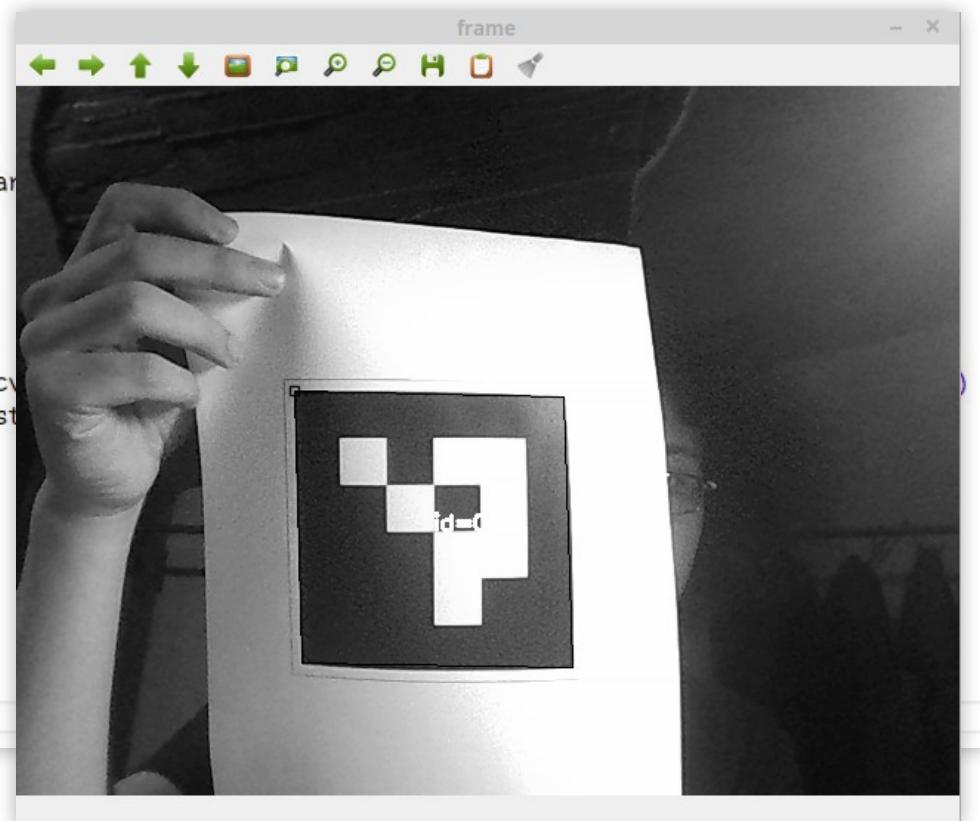
- Detección de ArUco con OpenCV

```
arucoDetectV0.py > ...
1 import numpy as np
2 import cv2
3 import time
4
5 cap = cv2.VideoCapture(2, cv2.CAP_V4L2)
6
7 parameters = cv2.aruco.DetectorParameters_create()
8 dictionary = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
9
10 while(1):
11     ret, frame = cap.read()
12     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
13
14     markerCorners, markerIds, rejectedCandidates = cv2.aruco.detectMarkers(gray, dictionary, parameters=parameters)
15     frameDrawn = cv2.aruco.drawDetectedMarkers(undistorted, markerCorners, markerIds)
16
17     cv2.imshow('frame', frameDrawn)
18
19     if cv2.waitKey(1) & 0xFF == ord('q'):
20         break
21
22 cap.release()
23 cv2.destroyAllWindows()
24
```

Procedimientos y pruebas

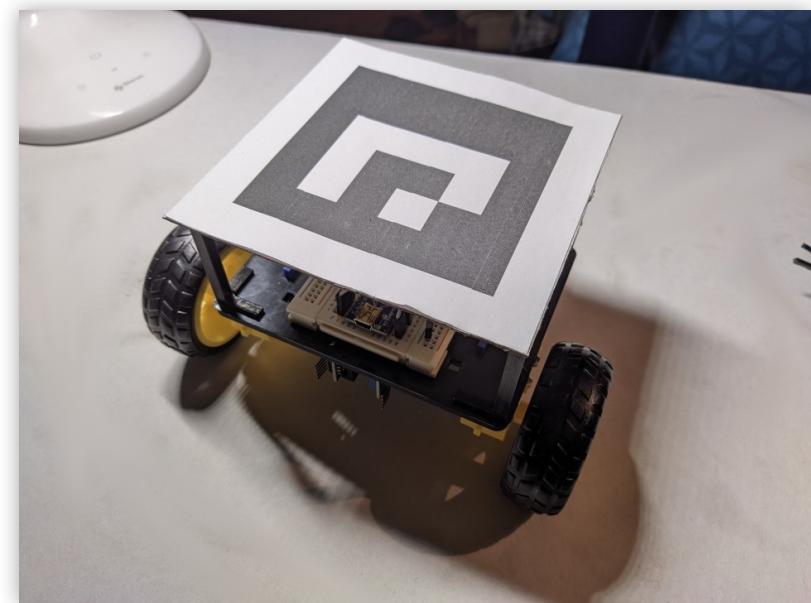
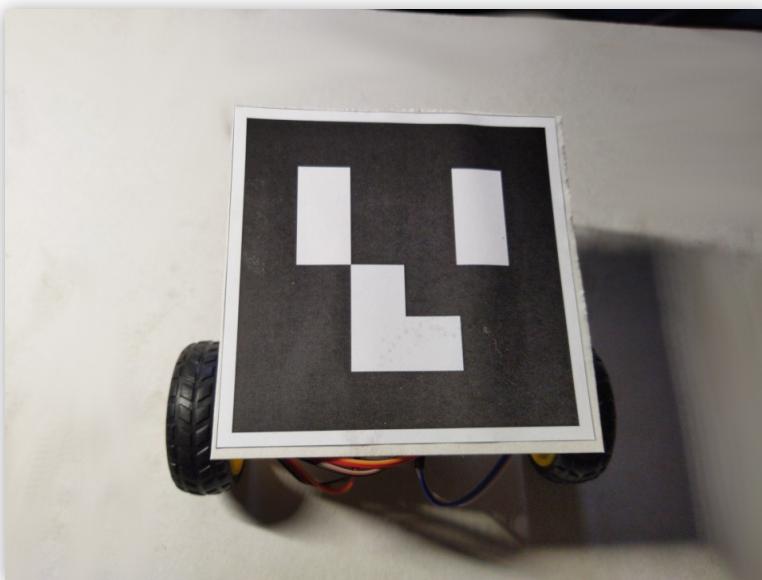
- Detección de ArUco con OpenCV

```
arucoDetectV0.py > ...
1 import numpy as np
2 import cv2
3 import time
4
5 cap = cv2.VideoCapture(2, cv2.CAP_V4L2)
6
7 parameters = cv2.aruco.DetectorParameters_create()
8 dictionary = cv2.aruco.getPredefinedDictionary(cv2.ar
9
10 while(1):
11     ret, frame = cap.read()
12     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
13
14     markerCorners, markerIds, rejectedCandidates = cv
15     frameDrawn = cv2.aruco.drawDetectedMarkers(undist
16
17     cv2.imshow('frame', frameDrawn)
18
19     if cv2.waitKey(1) & 0xFF == ord('q'):
20         break
21
22 cap.release()
23 cv2.destroyAllWindows()
24
```



Procedimientos y pruebas

- Detección de ArUco con OpenCV

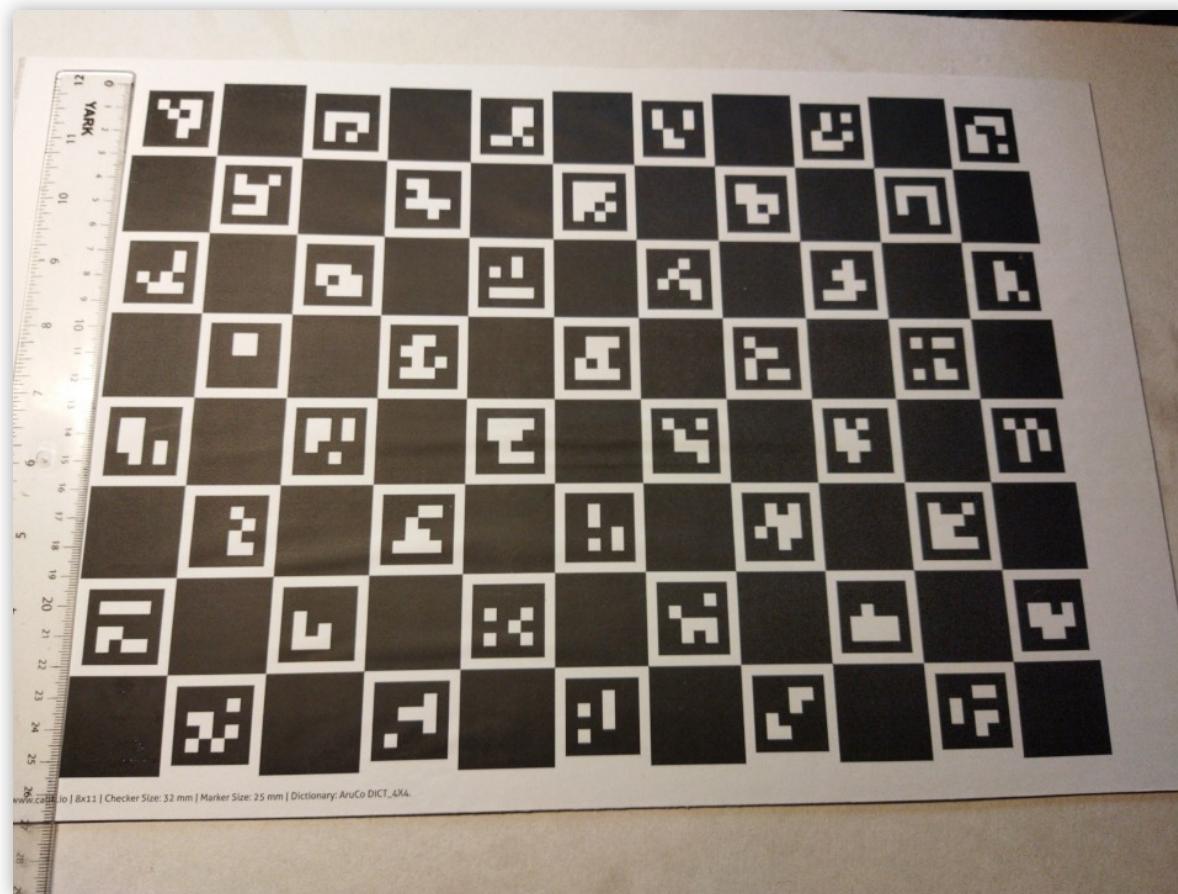


Procedimientos y pruebas

- Calibración de lente y cambio de perspectiva con tabla ChArUco

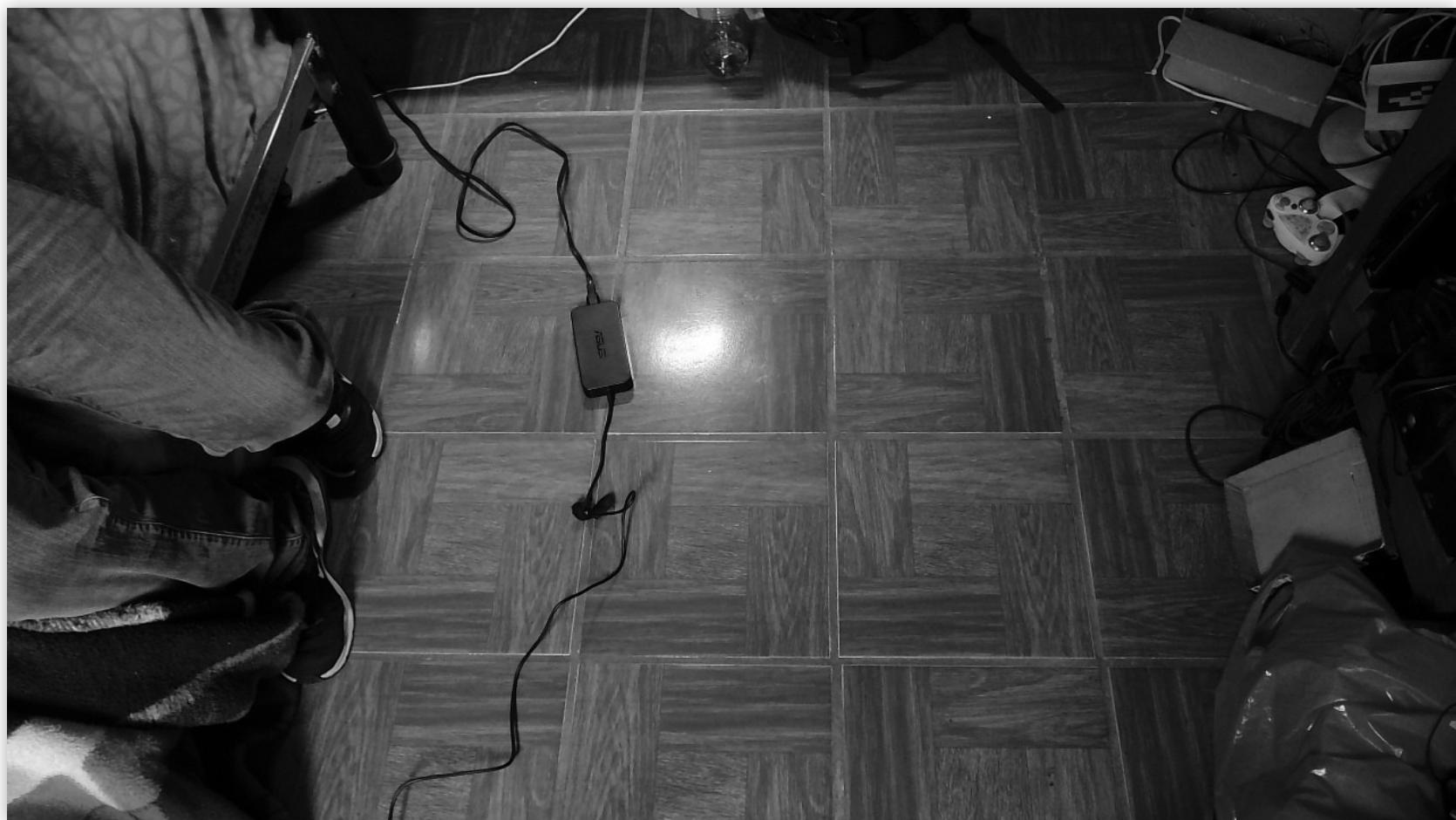
Procedimientos y pruebas

- Calibración de lente y cambio de perspectiva con tabla ChArUco



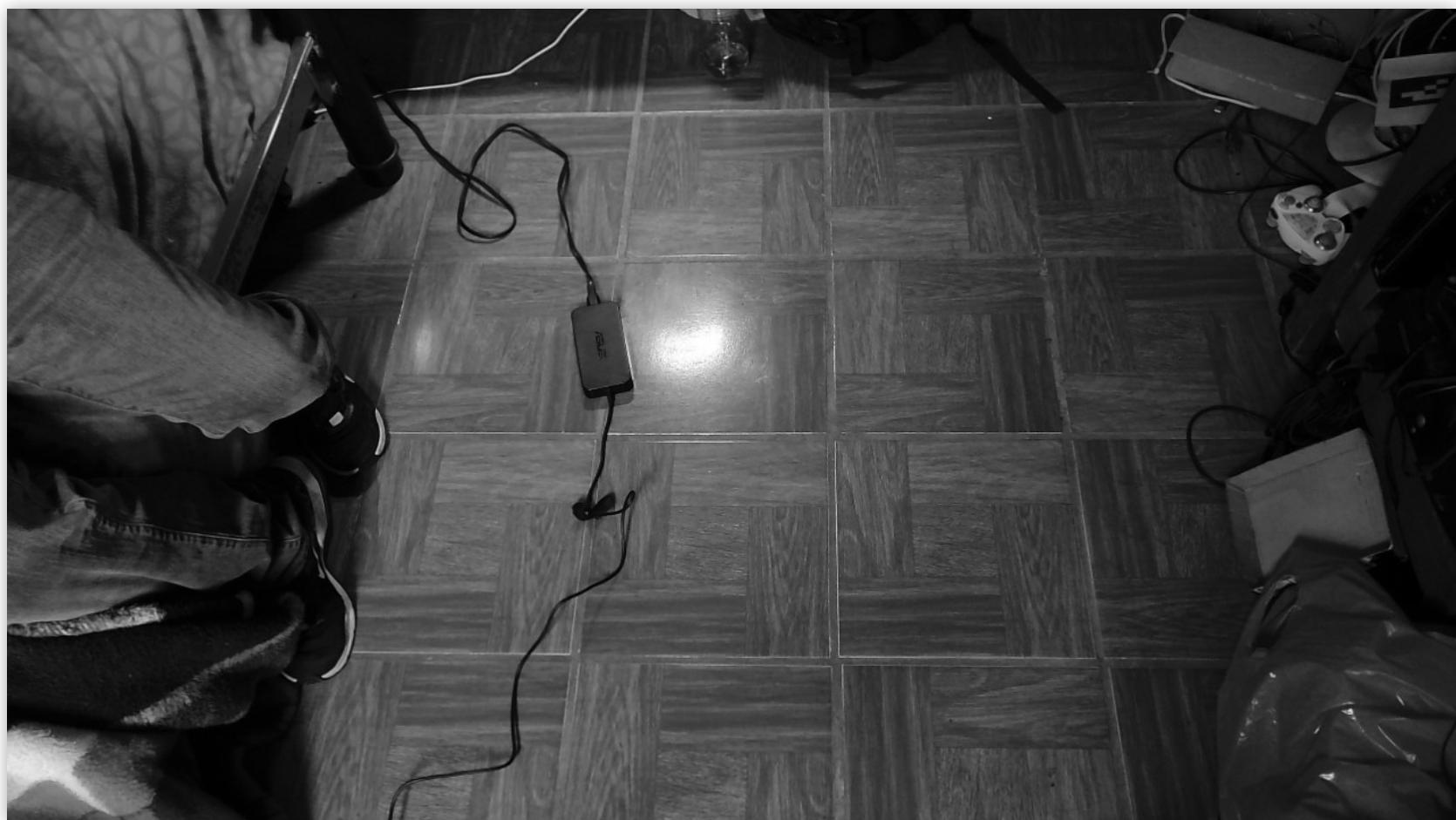
Procedimientos y pruebas

- Calibración de lente y cambio de perspectiva con tabla ChArUco



Procedimientos y pruebas

- Calibración de lente y cambio de perspectiva con tabla ChArUco



Procedimientos y pruebas

- Calibración de lente y cambio de perspectiva con tabla ChArUco

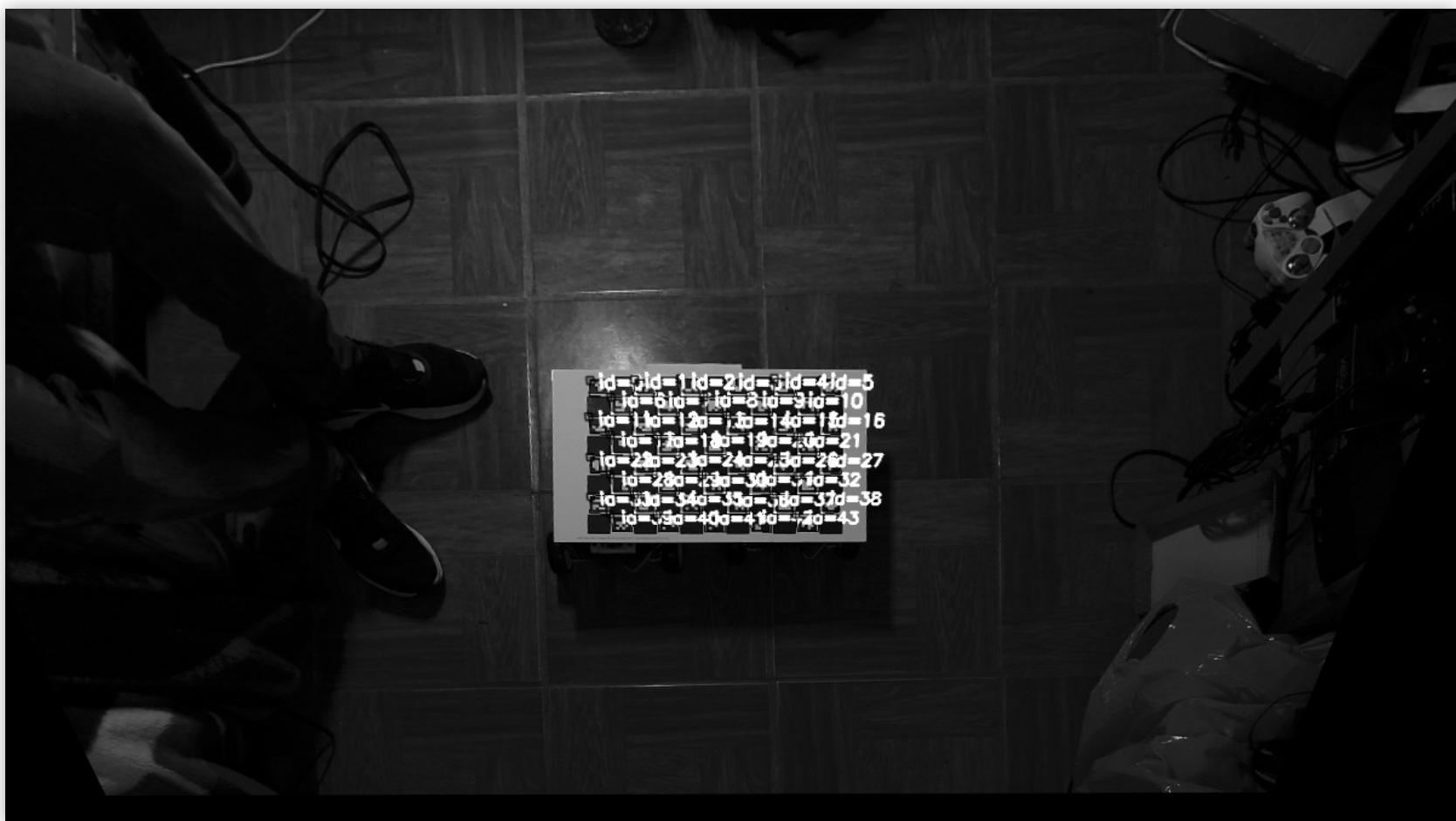
Procedimientos y pruebas

- Calibración de lente y cambio de perspectiva con tabla ChArUco



Procedimientos y pruebas

- Calibración de lente y cambio de perspectiva con tabla ChArUco



Procedimientos y pruebas

- Control de los robots por comandos inalámbricos

Procedimientos y pruebas

- Control de los robots por comandos inalámbricos

```
serialSender.py > ...
1 import serial
2 import time
3
4 ser = serial.Serial('/dev/ttyACM0', 230400)
5
6 time.sleep(2.5)
7
8 def robotControl(id, velocity, direction):
9     vel = velocity.to_bytes(2, byteorder='little', signed=True)
10    dir = direction.to_bytes(2, byteorder='little', signed=True)
11
12    ser.write([ord('R'),id,vel[0],vel[1],dir[0],dir[1]])
13
14 while True:
15    robotControl(3, 0, 100)
16    time.sleep(0.050)
17    robotControl(3, 0, 0)
18    time.sleep(0.050)
19
```

Procedimientos y pruebas

- Control PID con retroalimentación de la cámara

Procedimientos y pruebas

- Control PID con retroalimentación de la cámara

```
class PID:  
    sumError = 0  
    pastError = 0  
  
    def __init__(self, KP=0.0, KI=0.0, KD=0.0):  
        self.KP = KP  
        self.KI = KI  
        self.KD = KD  
  
    def control(self, error):  
        self.sumError += error  
        self.sumError = max(min(150, self.sumError), -150)  
        if error == 0.0: self.sumError = 0.0  
        print(self.sumError)  
  
        if self.pastError == 0: self.pastError = error  
  
        P = self.KP * error  
        I = self.KI * self.sumError  
        D = self.KD * (error - self.pastError)  
  
        self.pastError = error  
  
        return P + I + D
```

Procedimientos y pruebas

- Control PID con retroalimentación de la cámara

```
robotDirPID = [None, \
    PID(KP=0.15, KI=0.165, KD=0.0), \
    PID(KP=0.15, KI=0.165, KD=0.0), \
    PID(KP=0.15, KI=0.165, KD=0.0)]
def robotControl(markerCorners, markerIds):
    if markerIds is not None:
        line = [markerCorners[0][0][3], markerCorners[0][0][2]]
        angle = math.atan2(line[1][1] - line[0][1], line[1][0] - line[0][0])
        position = [line[0][0] + ((line[1][0] - line[0][0])/2), line[0][1] + ((line[1][1] - line[0][1])/2)]
        print(position)

        angleDelta = smallestSignedAngleBetween(angle, PI/2)
        if (abs(angleDelta) > (3 * PI/180)):      # Allow 3 degrees of error
            dirError = angleDelta * 128          # make the value larger so the PID work better
        else:
            dirError = 0.0

        print(angle * (180/PI))
        print(angleDelta * (180/PI))

        dirControl = int(robotDirPID[1].control(dirError))

        if dirControl > 127: dirControl = 127
        print(dirControl)
        robotSendMove(1, 0, dirControl)
```

Procedimientos y pruebas

- Control PID con retroalimentación de la cámara

```
robotDirPID = [None, \
    PID(KP=0.15, KI=0.165, KD=0.0), \
    PID(KP=0.15, KI=0.165, KD=0.0), \
    PID(KP=0.15, KI=0.165, KD=0.0)]
1 def robotControl(markerCorners, markerIds):
1     if markerIds is not None:
1         line = [markerCorners[0][0][3], markerCorners[0][0][2]]
1         angle = math.atan2(line[1][1] - line[0][1], line[1][0] - line[0][0])
1         position = [line[0][0] + ((line[1][0] - line[0][0])/2), line[0][1] + ((line[1][1] - line[0][1])/2)]
1         print(position)
1
1         angleDelta = smallestSignedAngleBetween(angle, PI/2)
1         if (abs(angleDelta) > (3 * PI/180)):      # Allow 3 degrees of error
1             dirError = angleDelta * 128          # make the value larger so the PID work better
1         else:
1             dirError = 0.0
1
1         print(angle * (180/PI))
1         print(angleDelta * (180/PI))
1
1         dirControl = int(robotDirPID[1].control(dirError))
1
1         if dirControl > 127: dirControl = 127
1         print(dirControl)
1         robotSendMove(1, 0, dirControl)
```

```
16     # https://stackoverflow.com/a/1878936
17     PI = math.pi
18     TAU = PI * 2
19     def smallestSignedAngleBetween(x, y):
20         a = (x - y) % TAU
21         b = (y - x) % TAU
22         return -a if a < b else b
```

Procedimientos y pruebas

- Control PID en los robots con setpoint en pulsos de encoder

Procedimientos y pruebas

- Control PID en los robots con setpoint en pulsos de encoder
 - Bibliotecas FastPID.h y Encoder.h

Procedimientos y pruebas

- Control PID en los robots con setpoint en pulsos de encoder

```
void loop()
{
    EncMA_count = EncMA.read();
    EncMB_count = EncMB.read();

    EncMA_countSetpoint = 120;
    EncMB_countSetpoint = 120;

    controlProcess();
}
```

```
93  const uint32_t periodPID = 1000000 / hz;
94  uint32_t lastTimePID = 0;
95  void inline controlProcess()
96  {
97      if((micros() - lastTimePID) > periodPID)
98      {
99          int16_t velA;
100         int16_t velB;
101
102         lastTimePID = micros();
103
104         int16_t errorA = EncMA_count - EncMA_countSetpoint;
105         int16_t errorB = EncMB_count - EncMB_countSetpoint;
106         if (abs(errorA) ≤ 1) {
107             PIDMA.clear();
108             velA = 0;
109         }
110         else {
111             velA = PIDMA.step(EncMA_countSetpoint, EncMA_count);
112         }
113
114         if (abs(errorB) ≤ 1) {
115             PIDMB.clear();
116             velB = 0;
117         }
118         else {
119             velB = PIDMB.step(EncMB_countSetpoint, EncMB_count);
120         }
121
122         motorControl(velA, velB);
123     }
124 }
125 }
```

Procedimientos y pruebas

- Doble PID por motor: añadiendo un control de velocidad

Procedimientos y pruebas

- Doble PID por motor: añadiendo un control de velocidad

Procedimientos y pruebas

- Doble PID por motor: añadiendo un control de velocidad
 - Se necesita conocer la velocidad...

Procedimientos y pruebas

- Doble PID por motor: añadiendo un control de velocidad

```
float hz=60.0;
float kp=2.0, ki=0.05, kd=0.1;
float kpS=0.04, kis=0.1, kds=0.0;

FastPID PIDposMA (kp, ki, kd, hz, 16, true);
FastPID PIDposMB (kp, ki, kd, hz, 16, true);

FastPID PIDspdMA (kpS, kis, kds, hz, 16, true);
FastPID PIDspdMB (kpS, kis, kds, hz, 16, true);
```

Procedimientos y pruebas

- Doble PID por motor: añadiendo un control de velocidad

```
float hz=60.0;
float kp=2.0, ki=0.05, kd=0.
float kpS=0.04, kis=0.1, kds

FastPID PIDposMA (kp, ki, kd
FastPID PIDposMB (kp, ki, kd

FastPID PIDspdMA (kps, kis,
FastPID PIDspdMB (kps, kis, 117 void inline encProcess()
118 {
119     EncMA_count = EncMA.read();
120     EncMB_count = EncMB.read();
121     uint32_t currTimePulseA = micros();
122     uint32_t currTimePulseB = micros();

// Calculo para Encoder A
123     if (EncMA_count != EncMA_prevCount)
124     {
125         uint32_t deltaTimeA = currTimePulseA - lastTimePulseA;
126         if (deltaTimeA > 1000)
127         {
128             if (EncMA_count > EncMA_prevCount)
129             {
130                 deltaArrayA[indexArrayA] = deltaTimeA;
131             }
132             else
133             {
134                 deltaArrayA[indexArrayA] = -(deltaTimeA);
135             }
136             indexArrayA++;
137             if (indexArrayA > (sizeof(deltaArrayA) / sizeof(uint32_t))) indexArrayA = 0;
138         }
139         EncMA_prevCount = EncMA_count;
140         lastTimePulseA = currTimePulseA;
141     }
142 }
143 }
```

Procedimientos y pruebas

- Doble PID por motor: añadiendo un control de velocidad

```
const uint32_t periodStop = 50000;
void inline spdCalcProcess()
{
    uint32_t currTimeCalc = lastTimeControl;

    // Calculo para Encoder A
    if((currTimeCalc - lastTimePulseA) > periodStop)
    {
        for (uint8_t i = 0; i < (sizeof(deltaArrayA) / sizeof(uint32_t)); i++) TimePulseA;
        {
            deltaArrayA[i] = 0;
        }
        avgSpeedA = 0;
    }
    else
    {
        int32_t avgTimeA = 0;
        for (uint8_t i = 0; i < (sizeof(deltaArrayA) / sizeof(uint32_t)); i++) );
        {
            avgTimeA += deltaArrayA[i];
        }
        avgSpeedA = 3145728 / (avgTimeA | 1);
    }
}
```

Procedimientos y pruebas

- Doble PID por motor: añadiendo un control de velocidad

```
const uint32_t periodStop = 50000;
void inline spdCalcProcess()
{
    uint32_t currTimeCalc = lastTimeCalc;

    // Calculo para Encoder A
    if((currTimeCalc - lastTimePulseA) > periodStop)
    {
        for (uint8_t i = 0; i < (sizeA - 1); i++)
        {
            deltaArrayA[i] = 0;
        }
        avgSpeedA = 0;
    }
    else
    {
        int32_t avgTimeA = 0;
        for (uint8_t i = 0; i < (sizeA - 1); i++)
        {
            avgTimeA += deltaArrayA[i];
        }
        avgSpeedA = 3145728 / (avgTimeA + 1);
    }
}

286 const int16_t motorMaxSpd = 150;
287 void inline PIDSpdProcess()
288 {
289     int16_t accelA = PIDspdMA.step(spdSetpointA, avgSpeedA);
290     int16_t accelB = PIDspdMB.step(spdSetpointB, avgSpeedB);
291
292     // motorSpdA = spdSetpointA + accelA;
293     // motorSpdB = spdSetpointB + accelB;
294
295     motorSpdA += accelA;
296     motorSpdB += accelB;
297
298     if (motorSpdA > motorMaxSpd) motorSpdA = motorMaxSpd;
299     else if (motorSpdA < -motorMaxSpd) motorSpdA = -motorMaxSpd;
300
301     if (motorSpdB > motorMaxSpd) motorSpdB = motorMaxSpd;
302     else if (motorSpdB < -motorMaxSpd) motorSpdB = -motorMaxSpd;
303
304     motorControl(motorSpdA, motorSpdB);
305 }
```

Procedimientos y pruebas

- Movimiento de los robots con el ratón

Procedimientos y pruebas

- Movimiento de los robots con el ratón
 - De pixeles, radianes y milímetros a pulsos de encoder

Procedimientos y pruebas

- Movimiento de los robots con el ratón
 - De pixeles, radianes y milímetros a pulsos de encoder

```
unitPerMillimeter = 60.0 / 97.5
pulsesPerRevolution = 60
wheelDiameter = 66.5 * unitPerMillimeter
distanceBetweenWheel = 165.0 * unitPerMillimeter
pulsesPerRad = pulsesPerRevolution / TAU
pulsesPerUnit = pulsesPerRevolution / (wheelDiameter * PI)
pulsesPerRadRobot = pulsesPerRad * (distanceBetweenWheel / wheelDiameter)
```

Procedimientos y pruebas

- Movimiento de los robots con el ratón

```
while(1):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # undistorted = cv2.undistort(gray, cameraMatrix, distCoeffs, None, None)
    warped = cv2.warpPerspective(gray, perspectiveMatrix, (1200, 800))

    markerCorners, markerIds, rejectedCandidates = cv2.aruco.detectMarkers(warped)

    robotGetPositions(markerCorners, markerIds)

    frameDrawn = cv2.aruco.drawDetectedMarkers(warped, markerCorners, markerIds)
```

Procedimientos y pruebas

- Movimiento de los robots con el ratón

```
while(1):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # undistorted = cv2.undistort(gray, cameraMatrix, distCoeffs, None, None)
    warped = cv2.warpPerspective(gray, perspectiveMatrix, (1200, 800))

    markerCorners, markerIds, rejectedCandidates = cv2.aruco.detectMarkers(warped)

def robotGetPositions(markerCorners, markerIds):
    global robotPositions
    if markerIds is not None:
        for i in range(len(markerIds)):
            # We only use robots with id 1,2,3
            if markerIds[i][0] > 0 and markerIds[i][0] ≤ 3:
                line = [markerCorners[i][0][3], markerCorners[i][0][2]]

                angle = math.atan2(line[1][1] - line[0][1], line[1][0] - line[0][0])
                position = [line[0][0] + ((line[1][0] - line[0][0])/2), line[0][1] + ((line[1][1] - line[0][1])/2)]

                robotPositionsLock.acquire()
                robotPositions[markerIds[i][0]] = [position, angle]
                robotMarkerCorners[markerIds[i][0]] = markerCorners[i][0]
                robotPositionsLock.release()
```

Procedimientos y pruebas

- Movimiento de los robots con el ratón

```
cv2.setMouseCallback('frame', mouseCallback)

robotSelected = 0
def mouseCallback(event,x,y,flags,params):
    global robotPositionsLock, robotMarkerCorners, robotSelected
    if event == cv2.EVENT_RBUTTONDOWN:
        if robotSelected != 0:
            process = threading.Thread(target=robotControl, args=(robotSelected, x, y,))
            process.start()
            print(x, y)
    elif event == cv2.EVENT_LBUTTONDOWN:
        robotPositionsLock.acquire()
        corners = robotMarkerCorners
        robotPositionsLock.release()
        # print(corners)
        for i in range(len(corners)):
            if corners[i] is not None:
                square = np.array(corners[i], np.int32)
                collision = cv2.pointPolygonTest(square, [x, y], False)
                if collision > 0:
                    print(i,[x, y], square, collision)
                    robotSelected = i
                    return
    robotSelected = 0
```

Procedimientos y pruebas

- Movimiento de los robots con el ratón

```
cv2.setMouseCallback('frame', mouseCallback)

robotSelected = 0
def mouseCallback(event, x, y, flags, param):
    global robotSelected
    if event == cv2.EVENT_LBUTTONDOWN:
        robotPositionsLock.acquire()
        position = robotPositions[robotSelected]
        robotPositionsLock.release()

    elif event == cv2.EVENT_MOUSEMOVE:
        robotSendRotate(robotSelected, angleDelta)
        time.sleep(2)
        robotSendTranslate(robotSelected, distance)

        collision = cv2.pointPolygonTest(square, [x, y], False)
        if collision > 0:
            print(i, [x, y], square, collision)
            robotSelected = i
            return

    robotSelected = 0
```

Resultados

Trabajos a Futuro

Trabajos a Futuro

- Mejorar el diseño del kit de robot diferencial

Conclusiones