Proyecto Final de Robótica

Victoria Medina, Juan Pablo Castillo, Rodrigo De Carcer e Isabel Díaz Miranda

December 20, 2017

Abstract

Este proyecto consiste en el diseño e implementación de varios componentes útiles para la conducción autónoma del robot AutoNOMOs. En donde sea relevante se utilizarán los parámetros del proyecto anterior. Los componentes del proyecto son:

- Estimación de estado propio: En base a una entrada de visión se estima en qué carril está el robot utilizando un filtro de histogramas.

- Estimación de estado de un obstáculo móvil: En base a una entrada de una nube de puntos de LiDAR se estima la velocidad a la que se mueve un objeto externo con respecto al robot.

- Movimiento para seguir al obstáculo móvil: Basádose en el controlador realizado en el proyecto anterior y en la estimación de estado del obstáculo móvil del componente anterior, se genera la velocidad y dirección de movimiento del robot para seguir al obstáculo.

1 Introducción

Los robots que se desarrollan y utilizan dentro de la ingeniería, la industria, y más recientemente en otras áreas, poseen un complejo software que les permite realizar sus funciones. Este mismo software es el que permite a los ingenieros y algunos técnicos desarrollar nuevas capacidades para los robots, así como resolver muchas de las limitantes que éstos poseen. Antes del año 2007, no había un consenso o una base sólida de software que los robots pudieran utilizar; dado lo anterior, la Universidad de Stanford dentro de sus proyectos como el STanford Artificial Intelligence Robot (SATIR) y el programa Personal Robotics (PR) desarrollaron marcos de trabajo prototipo que más adelante, al concretarse, se volvieron el Robot Operating System (ROS).

Willow Garage, una incubadora de robots en conjunto con múltiples investigadores, proveyó de recursos significativos para crear mejores implementaciones de los prototipos desarrollados por la Universidad de Stanford para sus robots. Al terminar el proyecto de Willow Garage, ROS nació en 2007 como software de licencia abierta para todos los usuarios y fue, gradualmente, utilizado por muchas más plataformas de robots en la comunidad de investigadores y desarrolladores.

Actualmente, ROS es un sistema que cuenta con más de 10,000 usuarios a lo largo del mundo, y está dentro de robots que van desde los pequeños hasta los grandes conjuntos industriales. El presente proyecto sirve como introducción a lo que es ROS y sus fundamentos, para aplicarlos en la construcción de nodos con capacidad de publicar y suscribirse a tópicos.

2 Marco Teórico

2.1 Representación de la pose del obstáculo con respecto al robot en función de las lecturas del LiDAR

Para esta representación se usará la dirección del eje z que obedece la regla de la mano derecha y forma un marco de coordenadas diestro. El punto P representa el obstáculo como se observa en la figura 1, el eje A corresponde al del robot.

La pose del obstáculo se puede representar usando el Teorema de Rotación de Euler, éste dice que cualquier rotación puede ser considerada como la secuencia de rotaciones en diferentes ejes de coordenadas.

Se empieza considerando un solo eje de coordenadas, como se observa en la figura 2. Es importante señalar que la operación de rotación no es conmutativa, por lo que el orden en que son aplicadas es muy importante.

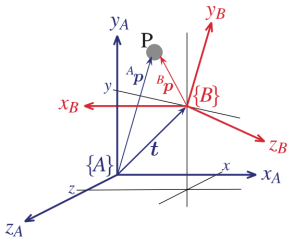

Figure 1: Dos marcos de coordenadas 3D A y B. B se gira y se observa con respecto a A

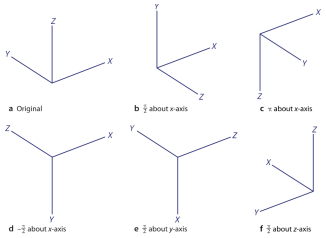

Figure 2: Rotación de un eje de coordenadas en 3 dimensiones

Podemos representar la orientación de un marco de coordenadas, correspondientes al obstáculo y obtenidas a través de las lecturas del LiDAR, por sus vectores unitarios expresados en términos del marco de coordenadas de referencia. Cada vector unitario tiene tres elementos y forman las columnas de una matriz ortonormal 3x3, como se observa en la figura 3. Esta gira un vector definido con respecto al eje B a un vector con respecto a A.

La matriz R pertenece al grupo especial ortogonal de dimensión 3 o R $\in$ SO (3) $\subset$ R3 $\times$ 3. Las matrices de rotación ortonormal para la rotación de θ sobre los ejes x, y y z se observan en la figura 4.

Para representar la traslación se puede utilizar la matriz de transformación homogénea que se observa en la figura 5. El vector de traslación entre el origen de los ejes de coordenadas es t y el cambio de orientación está representado por una submatriz ortonormal 3x3 R.

2.2 Propuesta de estrategia de control para que el robot iguale la velocidad del obstáulo móvil una vez que cuenta con la estimación de estado del mismo

Como estamos siguiendo un obstáculo móvil, dependemos de una pose dinámica. Por tanto, para igualar su velocidad, se igualará la pose del robot a la del obstáculo móvil a través del siguiente controlador.

La pose deseada es (x*, y*, θ*). Se puede representar la pose como en la figura 6. Después transformamos las ecuaciones en coordenadas polares usando la notación de la figura 7. Este movimiento se observa en la figura 8.

Esto resulta en las ecuaciones de la figura 9 y se utilizan las ecuaciones de control lineal de la figura 10. El sistema de circuito cerrado de la figura 11 se mantiene estable mientras se cumplan las condiciones de la figura 12. La velocidad v siempre tiene una constante signo que depende del valor inicial de α.

$$\begin{pmatrix} {}^Ax \\ {}^Ay \\ {}^Az \end{pmatrix} = {}^A\boldsymbol{R}_B \begin{pmatrix} {}^Bx \\ {}^By \\ {}^Bz \end{pmatrix}$$

Figure 3: Matriz ortonormal

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$
$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$
$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 4: Ejes ortonormales en x,y,z

$$\begin{pmatrix} {}^A x \\ {}^A y \\ {}^A z \\ 1 \end{pmatrix} = \begin{pmatrix} {}^A \boldsymbol{R}_B & \boldsymbol{t} \\ \boldsymbol{0}_{1\times 3} & 1 \end{pmatrix} \begin{pmatrix} {}^B x \\ {}^B y \\ {}^A z \\ 1 \end{pmatrix}$$

Figure 5: Matriz de rotación y traslación

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \gamma \end{pmatrix}$$

Figure 6: Pose

$$\rho = \sqrt{\Delta_x^2 + \Delta_y^2}$$
$$\alpha = \tan^{-1}\frac{\Delta_y}{\Delta_x} - \theta$$
$$\beta = -\theta - \alpha$$

Figure 7: Coordenadas polares

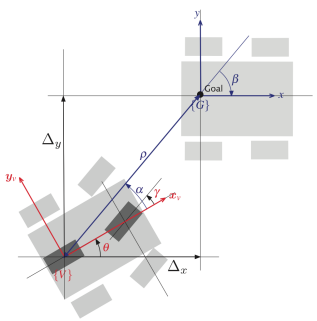

Figure 8: Notación de coordenadas polares para el modelo de bicicleta

$$\begin{pmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{pmatrix}=\begin{pmatrix}-\cos\alpha & 0\\ \frac{\sin\alpha}{\rho} & -1\\ -\frac{\sin\alpha}{\rho} & 0\end{pmatrix}\begin{pmatrix}v\\ \gamma\end{pmatrix},\ \text{if}\ \alpha\in\left(-\frac{\pi}{2},\frac{\pi}{2}\right]$$

Figure 9: Coordenadas en notación polar

$$v = k_\rho \rho$$
$$\gamma = k_\alpha \alpha + k_\beta \beta$$

Figure 10: Ecuaciones de control

2.3 Algoritmo que permite identificar rectas a partir de las lecturas del LiDAR

Un muy buen algoritmo para detección de bordes y rectas es el algoritmo de Hough. Este analiza pixel a pixel usando un arreglo acumulador donde se ecuentran los "votos" de cada pixel en la recta correspondiente.

Es decir, en lugar de representar las rectas como y = mx + b, estas se representan mediante parámetros de dos dimensiones (ρ, ϕ). Se observa la ecuación paramétrica en la figura 13. Así como el pseudocódigo en la figura 14.

A es el arreglo acumulador de votos del espacio (rho, theta) de dimensiones N-θ x N-ρ. El algoritmo de Hough, recorre cada punto (u, v) y recorre cada elemento A[i,j] con i dentro [1,N] identificando si satisface la ecuación de la figura 13.

2.4 Secuencia de operaciones de procesamiento de imágenes que permiten identificar los puntos que corresponden a los carriles de la carretera y su mapeo a coordenadas en el plano

En primer lugar, se eliminan los pixeles que están en el coche de fondo. Para poder describir las operaciones del procesamiento de imágenes en nuestro proyecto se emplearon métodos como lo son erode, dilate, filtro Canny así como la transformada de Hough.

```
    //Procesamiento básico antes de aplicar la tranformada de Hough.
cv::Mat kernel;
kernel = getStructuringElement(cv::MORPH_RECT,cv::Size(4,4));
cv::erode(image, image, kernel);
cv::erode(image, image, kernel);
cv::dilate(image, image, kernel);
cv::dilate(image, image, kernel);
```

El método “erode” es empleado dos veces, esto significa que la erosión (morfología) es aplicada en muchas iteraciones. La idea básica en la morfología binaria es probar una imagen con una forma predefinida simple sacando conclusiones sobre cómo esta forma encaja o no las formas en la imagen. Esta simple "sonda" se llamaelemento estructurante, y es en sí misma una imagen binaria (es decir, un subconjunto del espacio o de la cuadrícula). Posteriormente se emplea el método dilate que es una función que acepta una imagen en blanco y negro. Activa los píxeles que estaban

$$\begin{pmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{pmatrix}=\begin{pmatrix}-k_\rho\cos\alpha\\ k_\rho\sin\alpha-k_\alpha\alpha-k_\beta\beta\\ -k_\rho\sin\alpha\end{pmatrix}$$

Figure 11: Sistema de circuito cerrado

$$k_\rho > 0, k_\beta < 0, k_\alpha - k_\rho > 0$$

Figure 12: Condiciones de control

$$\rho = usin\theta + vcos\theta$$

Figure 13: Recta de forma paramétrica

cerca de los píxeles que estaban originalmente, de modo que los elementos de la imagen se vuelven más gruesos.

Por ejemplo: figura 15.

Por otro lado se emplea el filtro Canny:

Canny(image,dst,50,200,3)

Cuyo propósito es descubrir o detectar los bordes, para finalmente emplear la transformada de Hough que nos ayuda a detectar figuras o líneas en una imagen. En este caso usamos Hough Lines para guardar los valores de las líneas encontradas y como método de comprobación mostramos la imagen original y la imagen procesada.

```
//Aplicamos la transformada de Hough y guardamos los valores de las líneas.
HoughLinesP(dst,lines,1,CV_PI/180,50,50,10);

//Grafica cada una de las líneas generadas por la transformada de Hough.
for( size_t i = 0; i < lines.size(); i++ )
{
        cv::Vec4i l = lines[i];
        line( cdst, cv::Point(l[0], l[1]), cv::Point(l[2], l[3]),
}

//Muestra la imagen original de la bolsa y la imagen procesada.
cv::imshow("Bolsa", cdst);
cv::imshow("Procesamiento", image2);
```

2.5 Filtro de Kalman y como se utiliza en la estimación de estado del obstáulo móvil

El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman en 1960 que sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal, al igual que el observador de Luenberger, pero sirve además cuando el sistema está sometido a ruido blanco aditivo. La diferencia entre ambos es que en el observador de Luenberger, la ganancia K de realimentación del error debe ser elegida "a mano", mientras que el filtro de Kalman es capaz de escogerla de forma óptima cuando se conocen las varianzas de los ruidos que afectan al sistema. Si el movimiento (de un objeto) en una escena es continuo es, obviamente, posible hacer predicciones sobre él en cada instante basado en las trayectorias previas. El sistema físico se modela por un vector de estados x, llamado simplemente el estado, y un conjunto de ecuaciones llamado el modelo del sistema. El modelo del sistema es una ecuación de vectores que describe la evolución del estado con el tiempo.

Require: (u, v)
1: $\Delta_\theta \leftarrow \frac{\pi}{N_\theta}$
2: $\Delta_\rho \leftarrow \frac{2\rho_{max}}{N_\rho}$
3: **for** $(i \leftarrow 1; i <= N_\theta; i+ = \Delta_\theta)$ **do**
4: $\quad \theta \leftarrow i \times \Delta_\theta$
5: $\quad \rho \leftarrow usin\theta + vcos\theta$
6: $\quad j \leftarrow \lfloor \frac{\rho - \rho_{min}}{\Delta_\rho} + 1$
7: $\quad A[i, j] + +$
8: **end for**

Figure 14: Pseudocódigo del algoritmo de Hough

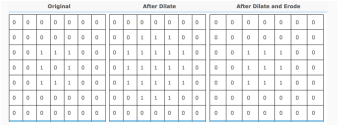

Figure 15: Efecto de los filtros

$$xk=\varphi k\text{-}1\ xk\text{-}1\ +\xi k\text{-}1$$

Figure 16: Modelo del sistema lineal

El tiempo de observación tiene la forma tk=t0+kΔT, k=0,1..., ΔT es el intervalo de muestreo y xk el estado x(tk). Vamos a suponer que ΔT es pequeño y que por tanto podemos utilizar un modelo del sistema lineal, como se ve en la figura 16.

donde ξk-1 es un vector aleatorio que modela el ruido aditivo. El subíndice k-1 en ϕ indica que la matriz de transición phi es (puede ser) una función del tiempo. El segundo ingrediente en la teoría de la estimación es el modelo de medida. Suponemos que en cada instante tk tenemos una observación ruidosa del vector de estados o al menos alguna de sus componentes mediante la siguiente relación en la figura 17.

zk es el vector de medidas tomadas en el instante tk. Hk es la llamada matriz de medidas y µk es un vector aleatorio que modela la incertidumbre asociada a las medidas. El estado de un sistema dinámico en el instante tk está descrito por un vector n-dimensional xk llamado vector de estados. La evolución del sistema se modela mediante la figura 18.

donde ϕk-1 es una matriz de tamaño nxn llamada matriz de transición de estados y ξk es un n-vector que modela el ruido asociado al modelo del sistema. Calcular el mejor estimador del sistema en el instante tk que notaremos xk teniendo en cuenta el estimador del estado predicho por el sistema en el instante tk-1, que obviamente notamos xk-1, y la medida realizada en el instante tk, que como sabemos hemos notado zk. Una aplicación común es la guía, navegación y control de vehículos, especialmente naves espaciales. Además el filtro es ampliamente usado en campos como procesamiento de señales y econometría.

2.6 Filtro de Histogramas y como se utiliza en la estimación de estado del robot con respecto a su carril

El histograma es aquella representación gráfica de estadísticas de diferentes tipos. La utilidad del histograma tiene que ver con la posibilidad de establecer de manera visual, ordenada y fácilmente comprensible todos los datos numéricos estadísticos que pueden tornarse difíciles de entender. Hay muchos tipos de histogramas y cada uno se ajusta a diferentes necesidades como también a diferentes tipos de información.

Los histogramas son utilizados siempre por la ciencia estadística. Su función es exponer gráficamente números, variables y cifras de modo que los resultados se visualicen más clara y ordenadamente. El histograma es siempre una representación en barras.

2.7 Método de aprendizaje por refuerzo para que el robot aprenda a mantenerse en su carril

El aprendizaje es el proceso de adquirir conocimientos, habilidades, actitudes o valores a través del estudio, la experiencia o la enseñanza. Un robot puede aprender sobre si mismo, sobre su entorno o sobre otros robots. El aprendizaje puede permitir que el robot realice mejor su tarea, ayudar a que el robot se adapte a los cambios en su entorno incluso puede simplificar el trabajo de programación del robot. Sin embargo, Russell y Norvig definen cuatro subcampos de aprendizaje:

$$zk=Hk\ xk\ +\mu k$$

Figure 17: Relación del modelo para cada instante

$$x_k = \varphi_{k-1}\, x_{k-1} + \xi_{k-1}$$

Figure 18: Evolución del sistema

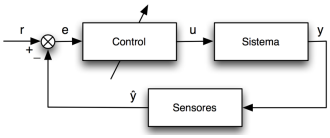

Figure 19: Aprendizaje por refuerzo

- Aprendizaje mediante ejemplos (inductivo, ID3, . . .).
- Redes de neuronal artificiales.
- Métodos estadísticos (bayesianos, evolutivos, . . .).
- Aprendizaje por refuerzo.

En este caso estudiaremos el aprendizaje por refuerzo, un tipo de aprendizaje que permite que los agentes aprendan a partir de la realimentación que reciben desde el entorno. Se basa en probar diferentes cosas y ver qué ocurre: Si las cosas van bien, tendemos a aplicar de nuevo ese comportamiento. Si las cosas van mal, tendemos a evitarlo. Con el aprendizaje por refuerzo, el robot prueba las diferentes acciones que tiene a su disposición en todos los estados en los que se encuentre y va registrando qué ocurre, como se ve en la figura 19.

Aprendizaje por refuerzo implica adquirir o incorporar nuevo conocimiento o habilidades para mejorar el rendimiento de un agente interactuando con el entorno que le rodea, no es supervisado. Además del propio agente y el entorno, podemos identificar hasta cuatro elementos principales más en los sistemas de aprendizaje por refuerzo:

- La política de control.
- La función de recompensa.
- La función de valor.

Los métodos de refuerzo especifican cómo debe cambiar la política el agente como se observa en la figura 20, resultado de la experiencia.

El objetivo del agente es maximizar la cantidad total de recompensa que obtiene durante el proceso. En nuestro caso nos funciona ya que nos facilita la programación del mismo, cabe recalcar que con este método no es necesario programar todas las posibles vertientes del mismo problema ya que el robot a través del conocimiento adquirido en situaciones previas es capaz de elegir una solución. Esto partiendo del supuesto de que el robot toma decisiones a ensayo y error, es decir, si la solución es la correcta tratara de seguir ese patrón, pero si o funciona desecha esta solución evitando el volver a usarla.

3 Descripción de la solución

3.1 Estimación de estado propio

Se declaran 7 estados, de los diferentes lugares donde puede encontrarse el robot como se observa en la figura 21.

Para identificar el estado en el que se encuentra el robot se usan dos criterios. El primer criterio está basado en las líneas de la carretera y el segundo criterio se basa en la dirección de la llanta.

La información de las líneas proviene del tópico /points/Center e informa si se observa una línea a la derecha, al centro o a la izquierda. Por lo que se declaran 4 casos que se observan el la

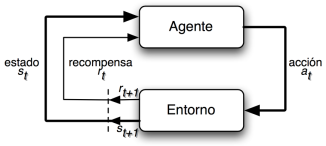

Figure 20: Política de agente

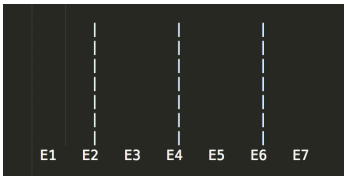

Figure 21: Estados posibles del robot

	Izquierda	Centro	Derecha
C1 –	NO	NO	SI
C2 –	NO	SI	NO
C3 –	SI	NO	SI
C4 –	SI	NO	NO

Figure 22: Casos de las rectas detectadas

figura 22.Después de declaran las probabilidades de los casos en los diferentes estados, figura 23.

Por otro lado, la información del steering proviene de Gazebo manualControl/steering. Este tiene valores del 0 al 180 y se clasifican como S1 = [0,60], S2 = [61, 120] y S3 = [121, 180]. Estos se dividen en probabilidades hacia donde se estima que el robot se moverá, como se ve en la figura 24.

Usando las tablas de las figuras 23 y 24 se formulan las tablas de probabilidad de todos los casos para determinar el estado actual del robot de la figura 25. Estas probabilidades son las que se usan para actualizar la llamada recursiva de códico en el anexo 6.1.

3.2 Estimación de estado de un obstáculo móvil

Tienes un tópico que existe en Gazebo que sustrae la información del LiDAR (distancia y ángulo a una nube de puntos). Después, con un cálculo trigonométrico se calcula la distancia del objeto al robot de las coordenadas x, y, ángulo en el que se encuentra. La velocidad es calculada a través de la diferencia en las distancias medidas en los momentos t y t-1.

Finalmente, se hace uso del filtro de Kalman para estimar el estado de los objetos que lo rodean. A través de estimaciones y mediciones recursivas hasta que la diferencia sea casi nula.

4 Experimentos

Para evaluar la primera parte del proyecto, se pondría al robot en la carretera de eagle knights en distintas posiciones iniciales dentro de la misma (fuera de la carretera, en el carril derecho o en el izquierdo). De esta manera se podría comprobar que el cálculo del estado propio es correcto.

Para evaluar la segunda parte del proyecto, podría montarse el código del archivo filtroKalman.cpp en el autoNOMOS y ponerlo en la carretera constuida en eagle knight junto con otro objeto o robot que represente al obstáculo del problema sugerido. De esta forma se vería los cálculos de la posición del segundo objeto respecto al robot autoNOMOS, e incluso si el obstáculo es móvil, posdíamos ver variaciones en la posición.

Para evaluar la tercera parte, se aprovecha el experimento anterior y, además de conocer la posición del segundo robot móvil, se agrega que el robot autoNOMOS seguiría al segundo robot de manera autónoma.

	E1	E2	E3	E4	E5	E6	E7
C1 = [	1	0	0	0	0	0	0]
C2 = [	0	1/3	0	1/3	0	1/3	0]
C3 = [	0	0	1/2	0	1/2	0	0]
C4 = [	0	0	0	0	0	0	1]

Figure 23: Probabilidades de los casos en los diferentes estados

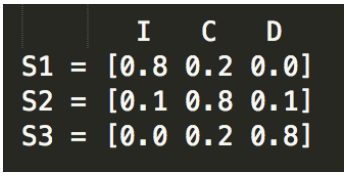

Figure 24: Probabilidades del steering

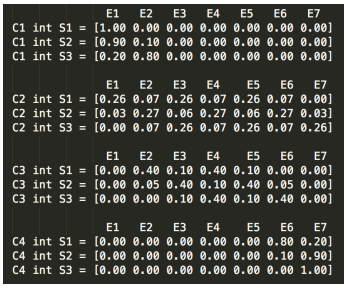

Figure 25: Tablas de probabilidad de todos los casos para determinar el estado del robot

5 Conclusiones

Por medio de este proyecto se aprendió emplear los conocimientos adquiridos durante el curso, específicamente el filtro de Kalman, el de Bayes, los histogramas y el algoritmo de Houghes. A lo largo del desarrollo de este se presentaron problemas con la subscripción y publicación en distintos nodos del robot. Así como el cálculo de las probabilidades del estado propio del robot.

Lamentablemente, por falta de tiempo y problemas no esperados en las partes uno y dos del proyecto no se logró realizar satisfactoriamente la parte 3 del proyecto. Esta no representa tanta dificultad, dado que se conoce la posición del obstáculo. La solución que se implementaría serái aplicar le modelo aplicado en el 2o proyecto, dado que se conoce el estado propio del robot y la ubicación del obstáculo.

Finalmente, los conocimientos que adquirimos en clase se sintetizaron en uno. Nos dimos cuenta de la utilidad de estos conocimientos por seeparado y su poder cuando se unen correctamente.

References

6 Anexo (códigos)

6.1 Estimación de estado propio

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include "std_msgs/MultiArrayLayout.h"
#include "std_msgs/MultiArrayDimension.h"
//#include "std_msgs/Int32MultiArray.h"
#include <iostream>
#include "std_msgs/Int16.h"
//#include <opencv2/highgui/highgui.hpp>
//#include "opencv2/imgproc/imgproc.hpp"
//#include <cv_bridge/cv_bridge.h>

#include <stdlib.h>
//#include <ionmanip>
#include <geometry_msgs/Pose2D.h>
#include <std_msgs/Int32MultiArray.h>
#include <math.h>
#include <string.h>
```

```
int Arr[90];
int der=2;
int izq=2;
int cen=2;
int caso=0;
int steer=0;
int steering=0;
int estado=0;
double gar=0.03;
double efr=0.05;
double sal=0.06;
double die=0.07;
double jp=0.1;
double sof=0.14;
double ro=0.2;
double mig=0.26;
double lui=0.27;
double mar=0.4;
double isa=0.8;
double vic=0.9;

/*double c1[]={1,0,0,0,0,0,0};
double c2[]={0,jp,0,jp,0};
double c3[]={0,0,isa,0,isa,0,0};
double c4[]={0,0,0,0,0,0,1};*/
double c1_s1[]={1,0,0,0,0,0};
double c1_s2[]={vic,jp,0,0,0,0,0};
double c1_s3[]={ro,isa,0,0,0,0,0};
double c2_s1[]={mig,die,mig,die,mig,die,0};
double c2_s2[]={gar,lui,sal,lui,sal,lui,gar};
double c2_s3[]={0,die,mig,die,mig,die,mig};
double c3_s1[]={0,mar,jp,mar,jp,0,0};
double c3_s2[]={0,efr,mar,jp,mar,efr,0};
double c3_s3[]={0,0,jp,mar,jp,mar,0};
double c4_s1[]={0,0,0,0,0,isa,ro};
double c4_s2[]={0,0,0,0,0,jp,vic};
double c4_s3[]={0,0,0,0,0,0,1};
double prob[]={sof,sof,sof,sof,sof,sof,sof};

using namespace std;
void arrayCallbackD(const std_msgs::Int32MultiArray& msg)
{
//ROS_INFO_STREAM("HOLA");
der=1;
}

void arrayCallbackI(const std_msgs::Int32MultiArray& msg)
{
//ROS_INFO_STREAM("HOLA1");
izq=1;
}

void arrayCallbackC(const std_msgs::Int32MultiArray& msg)
{
//ROS_INFO_STREAM("HOLA2");
cen=1;
```

```
}

void arrayCallbackS(std_msgs::Int16 msg){
        steering=msg.data;
ROS_INFO_STREAM("Steer"<<steering);
}

void multProb(double ant[], double caso[]){
        int n=7;
        for(int i=0; i<n; i++){
        ant[i]=ant[i]*caso[i];
        ROS_INFO_STREAM("Ant[i]"<<ant[i]);
        }
        }

int defEst(double arr[]){
        int resp=0;
        double max=-1;
        int pos=-1;
        for(int i=0; i<7; i++){
                if(arr[i]>max){
        //ROS_INFO_STREAM("arr[i] "<<arr[i]);
                max=arr[i];
                pos=i;
                }
        }
        resp=pos+1;
ROS_INFO_STREAM("pos "<<resp);
        return pos;
}

int main(int argc, char **argv)
{
        ros::init(argc, argv, "arraySubscriber");
        ROS_INFO_STREAM("Empieza el código");
        ros::NodeHandle n;

        ros::Subscriber sub1 = n.subscribe("points/right", 10, &arrayCallbackD);
        ros::Subscriber sub2 = n.subscribe("points/left", 10, &arrayCallbackI);
        ros::Subscriber sub3 = n.subscribe("points/center", 10, &arrayCallbackC);
        ros::Subscriber sub4 = n.subscribe("manual_control/steering", 10, &arrayCallbackS);

        while(ros::ok){

        if(izq==0&&cen==0&&der==1){
                caso=1;
        }
        else{
                if(izq==0&&cen==1&&der==0){
                        caso=2;
                }
                else{
                        if(izq==1&&cen==0&&der==1){
                                caso=3;
                        }
                        else{
                                if(izq==1&&cen==0&&der==0){
                                        caso=4;
```

```
                                        }
                                }
                        }
                }
        //ROS_INFO_STREAM(caso);

                if(steering>=0&&steering<=60){
                        steer=1;
                }
                else{
                        if(steering>=61&&steering<=120){
                                steer=2;
                        }
                        else{
                                if(steering>=121&&steering<=160){
                                        steer=3;
                                }
                        }
                }
        ROS_INFO_STREAM(steer);

                if(caso==1){
                        if(steer==1){
                                estado = 1;
                        }
                        if(steer==2){
                                estado = 1;
                        }
                        if(steer==3){
                                estado = 2;
                        }
                }
                else{
                        if(caso==2){
                                if(steer==1){
                                        estado = 2;
                                }
                                if(steer==2){
                                        estado = 3;
                                }
                                if(steer==3){
                                        estado = 3;
                                }
                        }
                        else{
                                if(caso==3){
                                        if(steer==1){
                                                estado = 4;
                                        }
                                        if(steer==2){
                                                estado = 5;
                                        }
                                        if(steer==3){
                                                estado = 6;
                                        }
                                }
                                else{
                                        if(caso==4){
```

```
                                        if(steer==1){
                                                estado = 6;
                                        }
                                        if(steer==2){
                                                estado = 7;
                                        }
                                        if(steer==3){
                                                estado = 7;
                                        }
                                }
                        }
                }
        }

                ROS_INFO_STREAM("Estado: " << estado);
                izq = 0;
                der = 0;
                cen = 0;

                ros::spinOnce();
        }
}
```

6.2 Estimación de estado de un obstáculo móvil

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "gazebo_msgs/LinkStates.h"
#include "std_msgs/Float32.h"
#include "std_msgs/String.h"
#include "sensor_msgs/LaserScan.h"

//Autores: Isabel Díaz, Rodrigo De Carcer, Victoria Medina y Juan Pablo Castillo.

//Definimos algunas variables auxiliares para el procesamiento.
//Rate de actualización.
double rate_hz = 10;
float PI = 3.14159265;
float angulo = 0, distancia = 0;
float x0= 0, y0= 0;
float aux = 0;

//Vectores para guardar la información recibida de los sensores.
float lecturaRadar[360];
float angulos[360];

//Arreglos en que guardamos los resultados del cálculo de la posición del segundo robot.
float robotPose[2];
float robotVelocity[2];

//Variables necesarias para utilizar el Filtro de Kalman.
float lastP = 0.1;
float R = 0.1;
float medidaAnteriorPos[2] = {0,0};
float medidaAnteriorVel[2] = {0,0};
```

```
/*Definimos las funciones que procesan la información recibida.
void lidarCallback(sensor_msgs::LaserScan);
void divideAngulos();
void infoOtroCoche();
void filtroKalman();*/

//Callback del subscriber.
void lidarCallback(sensor_msgs::LaserScan lidar_from_mini){
        for (int i = 0; i< 360; i++){
                lecturaRadar[i] = lidar_from_mini.ranges[i];
        }
}

//Función que obtiene la posición del otro robot dentro del ambiente.
void infoOtroCoche(){
        //Para cada uno de los puntos, si está dentro de un rango, lo tomamos en cuenta.
        for (int i = 0; i < 360; i++){
                if(lecturaRadar[i] > .4 && lecturaRadar[i] < 5){
                        angulo = angulos[i];
                        distancia = lecturaRadar[i];
                }
        }

        robotPose[0] = distancia*cos(angulo*180/PI);
        robotPose[1] = distancia*sin(angulo*180/PI);

        //La velocidad del otro robot es calculada en base a la distancia que recorrió de la últim
        y0 = (obsPose[0][1] - y0)/rate_hz;
        x0 = (obsPose[0][0] - x0)/rate_hz;

        robotVelocity[0] = x0;
        robotVelocity[1] = y0;
}

//Función que aplica el filtro de Kalman para la información obtenida en el método anterior.
void filtroKalman(){
        float medidaPos[2];
        float medidaVel[2];

        medidaPos[0] = medidaAnteriorPos[0];
        medidaPos[1] = medidaAnteriorPos[1];
        medidaVel[0] = medidaAnteriorVel[0];
        medidaVel[1] = medidaAnteriorVel[1];


        float auxPos[2];
        float auxVel[2];
        float actualP = lastP;
        float gKalman = actualP /(actualP + R);

        auxPos[0] = robotPose[0];
        auxPos[1] = robotPose[1];
        auxVel[0] = robotVelocity[0];
        auxVel[1] = robotVelocity[1];

        float x1, y1, nuevaY0, nuevaX0, nuevaP;
```

```
        nuevaP = (1-gKalman)*actualP;
        x1 = medidaPos[0] + gKalman*(auxPos[0] - medidaPos[0]);
        y1 = medidaPos[1] + gKalman*(auxPos[1] - medidaPos[1]);
        nuevaX0 = medidaVel[0] + gKalman*(auxVel[0] - medidaVel[0]);
        nuevaY0 = medidaVel[1] + gKalman*(auxVel[1] - medidaVel[1]);

        medidaAnteriorPos[0] = x1;
        medidaAnteriorPos[1] = y1;
        medidaAnteriorVel[0] = nuevaX0;
        medidaAnteriorVel[1] = nuevaY0;

        lastP = nuevaP;
}

int main(int argc, char **argv){
        ros::init(argc, argv, "Segundo ejercicio");
        ros::NodeHandle nh;

        ros::Subscriber getLidarInfo = nh.subscribe("/AutoNOMOS_mini_1/laser_scan", 1000, &lidarCa
        ros::Rate rate(rate_hz);

        robotVelocity[0] = x0;
        robotVelocity[1] = y0;

        for(int i = 0; i< 360; i++){
                angulos[i] = pos;
                pos += 0.0175019223243;
        }

        while(ros::ok()){
                infoOtroCoche();
                filtroKalman();

                ros::spinOnce();
                rate.sleep();
        }

        return 0;
}
```