Sistema de visión con AprilTag

1. Objetivo de la práctica

Comprender los principios de funcionamiento de los sistemas de visión artificial por reconocimiento de marcadores y distinguir los usos, ventajas y desventajas de ellos.

2. Metas

Para la realización de la práctica se deben cumplir las siguientes metas:

- Establecer las condiciones requeridas para la implementación de AprilTag en alguna plataforma (ROS, ViSP, Matlab)
- Realizar las configuraciones básicas del software AprilTag para el reconocimiento de los marcadores.
- Identificar y acceder a la información proporcionada por el software AprilTag para la detección de la posición y orientación de los marcadores que se encuentran dentro del campo de visión de la cámara.
- Adaptar la información obtenida mediante AprilTag para la aplicación particular que se le va a dar.

3. Antecedentes

Con la creciente evolución y mejora de los sistemas de procesamiento y de cómputo se ha ido incrementando también el uso de sistemas basados en visión artificial para la instrumentación de robots y ambientes inteligentes.

Las aplicaciones de la visión artificial son muy variadas, desde el reconocimiento de marcadores específicos, como lo que se realizará en esta práctica, hasta el reconocimiento de bordes, formas, objetos e incluso rostros. Para lograr realizar este reconocimiento de manera correcta se utilizan distintos algoritmos que pueden ser englobados en algún tipo de software especializado que facilite y permita el uso de estas funciones de manera general. Uno de estos programas que permiten el uso de visión artificial es AprilTag, que posibilita la identificación de marcadores (*fiducials*).

AprilTag es un software desarrollado por el *April Robotics Laboratory* de la Universidad de Michigan que permite la identificación de distintas familias de marcadores haciendo uso de un sistema de visión. Este sistema de detección permite obtener de forma precisa la posición y orientación en 3D de los diferentes marcadores con respecto al marco de referencia de la cámara. La información sobre este software puede encontrarse en el sitio web oficial del laboratorio (https://april.eecs.umich.edu/software/apriltag).

Este sistema permite la creación de diferentes tipos de marcadores de acuerdo con las necesidades de cada proyecto, sin embargo, se incluyen algunas familias de marcadores que resultan suficientes para la mayoría de las aplicaciones. En la Figura 1 se muestran algunas de las familias de marcadores que han sido creadas para su uso con el software de AprilTag, cabe mencionar que estas familias de marcadores han sido optimizadas para hacer más robusta la detección. Para esta práctica se utilizaron los marcadores de la familia Tag36h11.

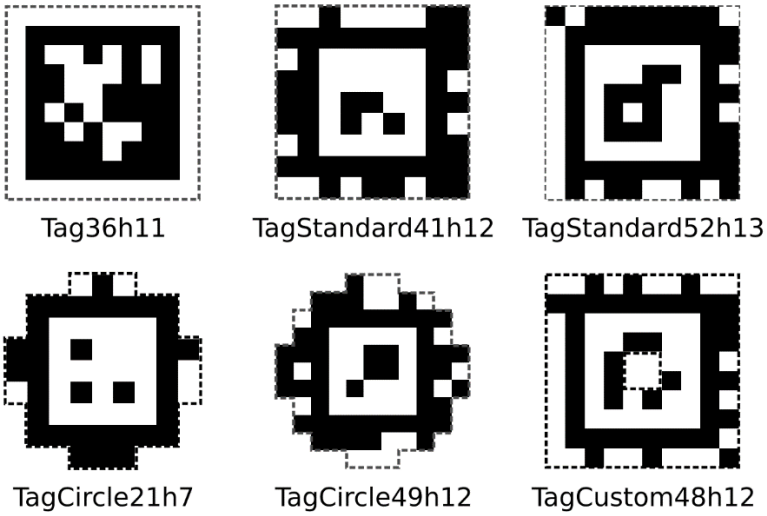


Figura 1. Familias de marcadores (fiducials).

Este tipo de marcadores presentan la ventaja de que pueden ser impresos fácilmente y colocados en cualquier elemento plano que se quiera identificar con el sistema de visión.

4. Conocimientos previos

Los conocimientos necesarios para la realización de la práctica:

- Conocimientos básicos de computación y programación.
- Conocimientos básicos de Linux.
- Conocimientos básicos de ROS.

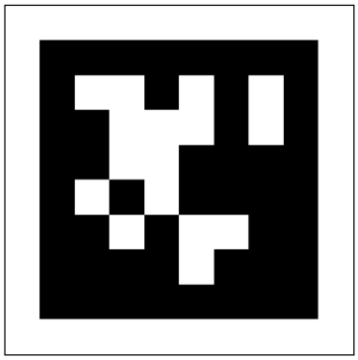
5. Materiales y Equipo

Para la realización de la práctica es necesario contar con lo siguiente:

- Una computadora con ROS. (1)
- Cámara web. (2)
- Conexión a internet.
- Marcadores (*fiducials*) impresos. (3)

(1)

(2)



(3)

6. Preparativos previos y recomendaciones

- Es recomendable crear un espacio de trabajo propio dentro de ROS.

7. Desarrollo de la práctica

a) Implementación de AprilTag con ROS

Transmitir la imagen de la cámara

Antes de poder utilizar AprilTag es necesario tener el acceso al video de las cámaras que utilizará el sistema, además de realizar una calibración de estas para asegurar que la información proporcionada por el sistema de visión sea lo más correcta posible.

Para poder transmitir la imagen de las cámaras en ROS y que pueda ser utilizado por el paquete de AprilTag, se propone utilizar un paquete llamado *video_stream_opencv*. Este paquete puede descargarse libremente de los paquetes de ROS y proporciona las herramientas para conectar la cámara con ROS y hacer la transmisión de video.

Al descargar el paquete se puede identificar un archivo con el nombre *camera.launch*, en este archivo es donde deberán realizarse las configuraciones para seleccionar la cámara que se va a utilizar, la resolución, los cuadros por segundo, el tamaño del buffer, entre otros. Los detalles de las configuraciones pueden ser consultados en la página del paquete *video_stream_opencv* (http://wiki.ros.org/video_stream_opencv).

Las principales configuraciones que deben realizarse dentro del archivo *camera.launch* son en las siguientes líneas de código:

```
<arg name="camera_name" value="camera_1" />
```

Define el nombre del nodo de ROS.

```
<arg name="video_stream_provider" value="1" />
```

Especifica el dispositivo de video que se está utilizando. Para seleccionar el valor correcto deberá conectarse la cámara a la computadora e identificar el número del dispositivo conectado. Una forma sencilla para mostrar los dispositivos de video conectados en Linux es utilizando la consola mediante el comando*: ls /dev/video** . De esta manera se mostrará la lista de los dispositivos de video conectados y podrá identificarse el número que corresponde a la cámara que se conectó.

```
<arg name="frame_id" value="cam1_frame" />
```

Establece el nombre del marco de referencia de la cámara.

```
<arg name="camera_info_url" value="file:///$(find video_stream_opencv)/confi
g/cam1_cal.yaml" />
```

Indica la ubicación del archivo de calibración de la cámara (más adelante se explica una manera de generar este archivo).

```
<arg name="visualize" value="false" />
```

Configura si se quiere visualizar el video en una ventana (true) o no (false).

Adicionalmente, en este mismo archivo se incluyen diversas propiedades que pueden configurarse dependiendo de la cámara que se esté utilizando, por ejemplo, la frecuencia (fps) o la resolución de la imagen.

Una vez que se ha realizado la configuración, resulta conveniente realizar una prueba habilitando la visualización para verificar que la transmisión del video se realiza correctamente.

Calibración de la cámara

Al utilizar una cámara para el sistema de visión se vuelve también necesario hacer la calibración de esta, de tal manera que los datos obtenidos sean lo más precisos posibles. Una forma sencilla de realizar la calibración es utilizando el paquete de ROS *camera_calibration* (http://wiki.ros.org/camera_calibration), el cual proporciona las herramientas para calibrar cámaras, tanto monoculares como estéreo. Este paquete permite generar el archivo de calibración con extensión *yaml*, que se solicitó en el archivo de configuración de la cámara del paquete *video_stream_opencv*, explicado anteriormente.

Como se utilizó una cámara monocular, se siguió el tutorial correspondiente para este tipo de cámaras, al cual se tiene acceso desde la página del paquete (http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration).

Este procedimiento requiere del uso de una cuadrícula como la que se muestra en la Figura 2, la cual debe moverse a diferentes posiciones, inclinaciones y distancias de la cámara para detectar la distorsión del lente y poder generar el archivo de calibración para cada una de las cámaras. En términos generales, el procedimiento para realizar la calibración consiste en, primero, ejecutar el nodo de calibración, una vez que ha iniciado y se ve la imagen de la cámara se debe desplazar la cuadrícula a diferentes posiciones y ángulos para capturar datos hasta que se activa el botón “CALIBRATE”.

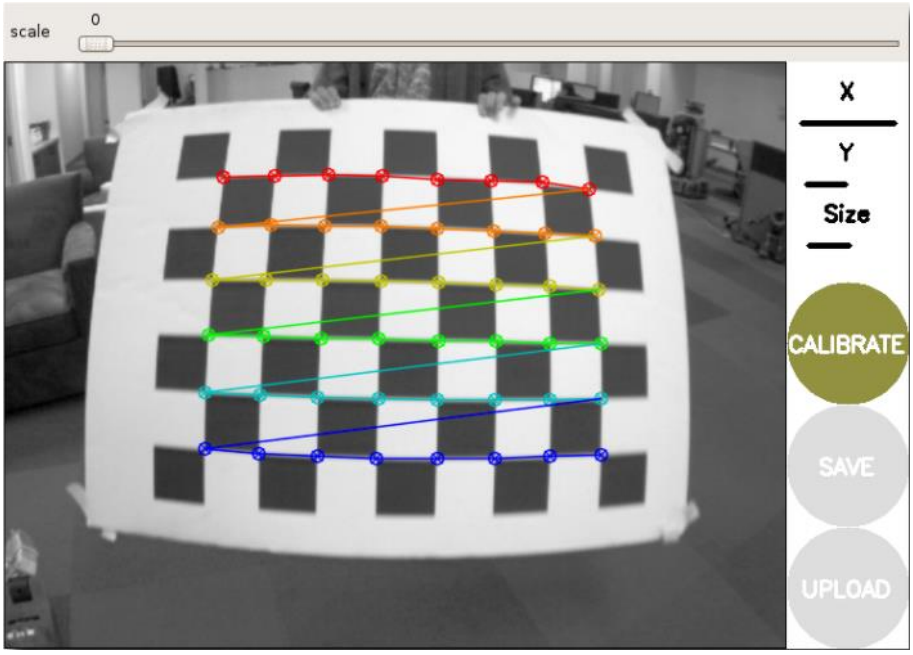


Figura 2. Calibración de una cámara con el paquete camera_calibration.

Posteriormente, se debe dar clic en este botón (CALIBRATE) para iniciar la calibración y esperar a que el programa termine de ejecutar. Después de terminar la calibración se puede cargar directamente a la cámara con el botón "UPLOAD", o bien, guardarla en un archivo con el botón "SAVE".

Rectificación de la imagen

Ya que se tiene el archivo de calibración es posible realizar la corrección a la imagen de la cámara, con la finalidad de utilizar la imagen rectificada para la detección de los marcadores. Una opción para realizar esta rectificación es mediante el uso del paquete *image_proc* (http://wiki.ros.org/image_proc). Este paquete recibe la información de la calibración y proporciona como salida la imagen rectificada.

Uso del paquete de AprilTag

Una vez que se tiene la imagen rectificada de la cámara, esta puede ser utilizada por el paquete *apriltag_ros* (http://wiki.ros.org/apriltag_ros) para identificar los marcadores. El paquete de *apriltag_ros* tiene como entradas la imagen rectificada y la información de la cámara, y como salidas los marcadores detectados, la imagen de la cámara con los marcadores que fueron identificados resaltados, y la publicación de las transformaciones en el módulo TF (Figura 3). Para su correcto funcionamiento es necesario asegurarse que los tópicos involucrados estén recibiendo la información que les corresponde.

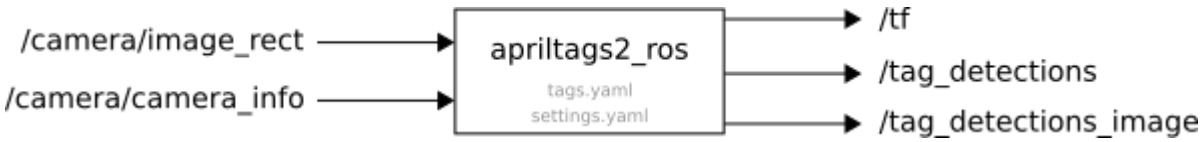


Figura 3. Tópicos de entrada y salida del paquete apriltag_ros.

Adicionalmente, se debe realizar la configuración localizada en el archivo *config/serttings.yaml*, especialmente en cuanto a seleccionar la familia de marcadores que se está utilizando, por ejemplo, la familia Tag36h11. También debe indicarse cuales números de marcadores se están utilizando, y deben ser identificados por el sistema en el archivo de configuración *config/tags.yaml*. Para ello existen dos posibilidades:

- **Identificación de marcadores individuales:** cuando se quiere detectar la posición y orientación de un marcador en específico bastará con indicar el id, el tamaño y el nombre en la sección denominada *standalone_tags*.

- **Identificación de un conjunto de marcadores:** cuando se quiere detectar un conjunto de marcadores que mantienen una relación fija entre ellos es necesario indicar todos los marcadores que conforman el grupo en la sección *tag_bundles*. Esta forma de detección puede mejorar la precisión al utilizar más de un marcador.

Un ejemplo de configuración de marcadores, en el que se muestran tanto marcadores individuales, como un conjunto de marcadores es el que se muestra a continuación:

```
standalone_tags:
  [
    {id: 0, size: 0.12, name: "cam1_t00"},
    {id: 1, size: 0.12, name: "cam1_t01"},
    {id: 2, size: 0.12, name: "cam1_t02"},
    {id: 3, size: 0.12, name: "cam1_t03"},
    {id: 8, size: 0.12, name: "cam1_t08"},
    {id: 9, size: 0.12, name: "cam1_t09"},
    {id: 10, size: 0.12, name: "cam1_t10"}
  ]

tag_bundles:
  [
    {
      name: 'lab',
      layout:
        [
          {id: 4, size: 0.12, x: -0.694, y: 0.483, z: 0.025, qw: 1.000, qx: 0.003, qy: 0.003, qz: 0.003},
          {id: 5, size: 0.12, x: 0.720, y: 0.482, z: 0.044, qw: 1.000, qx: 0.002, qy: -0.001, qz: -0.002},
          {id: 6, size: 0.12, x: 0.709, y: -0.432, z: 0.026, qw: 1.000, qx: -0.015, qy: 0.000, qz: -0.012},
          {id: 7, size: 0.12, x: -0.687, y: -0.443, z: 0.016, qw: 1.000, qx: -0.014, qy: 0.006, qz: 0.003}
        ]
    }
  ]
```

Después de realizar las configuraciones correspondientes se puede ejecutar el sistema de detección continua con el comando:

roslaunch apriltags2_ros continuous_detection.launch

En la página web del paquete se pueden consultar también los tutoriales para su uso, por ejemplo, para la detección de marcadores de forma continua: http://wiki.ros.org/apriltag_ros/Tutorials/Detection%20in%20a%20video%20stream

Uso de transformaciones desde TF

Como se observó anteriormente, el paquete *apriltag_ros* publica las transformaciones de los marcadores que fueron detectados mediante el paquete TF (http://wiki.ros.org/tf). Este es un paquete que permite mantener el seguimiento de múltiples marcos de referencia, permitiendo acceder a la información de las transformaciones entre ellos. Por ejemplo, utilizando el paquete *apriltag_ros* se publican las transformaciones entre el marco de referencia de la cámara y cada uno de los marcadores que fueron detectados por el programa AprilTag.

Para poder acceder a la información de estas transformaciones dentro de cualquier otro nodo de ROS es necesario generar un *Listener*, así mismo, para poder publicar cualquier transformación adicional será necesario generar un *Broadcaster*. La forma más fácil de aprender a implementar esto y familiarizarse con su uso es siguiendo los tutoriales proporcionados por el mismo paquete TF (http://wiki.ros.org/tf/Tutorials). Allí se proporciona la información para implementarlo tanto en C++ como en Python.

Una vez que se puede acceder a la información de las transformaciones mediante TF, se puede utilizar para cualquier aplicación que requiera de la posición y orientación de los marcadores. Por ejemplo, se puede colocar un marcador como origen del sistema de referencia y, posteriormente, obtener las transformaciones del resto de marcadores con respecto a este origen.

b) Implementación de AprilTag con ViSP

Si bien la implementación de AprilTag utilizando la paquetería de ROS resulta bastante útil, también existen otras alternativas para utilizar este software de detección de marcadores. Una posibilidad es hacer uso de alguna plataforma de visión artificial en la que esté incluido el funcionamiento del software de AprilTag. Por ejemplo, Visual Servoing Platform (ViSP) es una librería que incluye un conjunto de programas de software libre con funciones relacionadas con visión artificial y procesamiento de imágenes, entre ellos AprilTag. La librería está desarrollada en C++ y el grupo Inria Rainbow es quién lo desarrolla y le da mantenimiento.

Esta plataforma puede ser descargada directamente desde la página web de la plataforma (https://visp.inria.fr/), donde pueden encontrarse las versiones para los diferentes sistemas operativos incluidos Windows, Mac y Linux.

En la misma documentación de la librería se pueden encontrar tutoriales para realizar la instalación de ViSP dependiendo del sistema operativo que se esté utilizando. Para Windows, se puede realizar la instalación utilizando el programa Visual C++, tal como se explica en el tutorial: https://visp-doc.inria.fr/doxygen/visp-3.3.0/tutorial-install-win10-msvc16.html .

Tras realizar correctamente la instalación de ViSP se tendrá acceso a los diferentes programas incluidos en esta librería. Como primer paso, se recomienda realizar la calibración de la cámara siguiendo el tutorial: https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-calibration-intrinsic.html. Posteriormente, para hacer uso de AprilTag con ViSP también se cuenta con un tutorial (https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-detection-apriltag.html) en el que se explica la forma de implementarlo mediante unos ejemplos que detectan los marcadores, tanto en una imagen fija, como en la transmisión continua desde la cámara.

Una vez calibrada la cámara se puede probar el funcionamiento del software ejecutando los ejemplos que vienen en la instalación, en especial los que realizan la identificación de los marcadores que pueden localizarse (dependiendo de la instalación) en la dirección: C:\visp-ws\visp-build-vc16\tutorial\detection\tag. Aquí se incluyen dos programas de ejemplo, uno de ellos funciona con una sola imagen, identificando y marcando las posiciones y orientaciones de los Tags, mientras que el otro funciona en vivo actualizando de manera constante la imagen desde la cámara conectada al dispositivo. Éste último programa es el que se sugiere tomar como base para modificarlo y tener una versión propia que obtenga las posiciones de los marcadores con respecto a otro que se toma como referencia, y que sea capaz de enviar dicha información mediante el protocolo UDP para poder ser utilizada en otras aplicaciones, por ejemplo, en Matlab.

Para poder utilizar el programa con estos fines se puede hacer lo siguiente:

1. Se realiza el procedimiento para conexión con la cámara y para la configuración de los diferentes parámetros. En este caso se dejó la configuración para la conexión y obtención de las imágenes de la cámara que viene en el ejemplo.
2. Análisis de la imagen para identificar los marcadores y obtener sus posiciones. Para lograr esto se hizo uso de una función de la programación de AprilTag:

   ```
   detector.detect(I, tagSize, cam, cMo_vec);
   ```

   Con esta función el programa identifica las posiciones de los marcadores en tres dimensiones y las devuelve en forma de matrices homogéneas de transformación con respecto al sistema de referencia de la cámara, una para cada marcador encontrado. Éstas se almacenan en forma de arreglo en la variable denotada por cMO_vec, a la que se puede acceder posteriormente.

3. Para poder tener las coordenadas de los marcadores en función del sistema de referencia de otro marcador (que se define como origen del sistema de coordenadas global), se puede hacer una multiplicación de las matrices homogéneas de tal forma que se obtenga la transformación deseada. En este caso se obtuvo la matriz inversa de la matriz homogénea de transformación del marcador del origen con respecto a la cámara, de tal manera que la matriz de transformación de un marcador cualquiera con respecto al del marcador de origen puede obtenerse multiplicando dicha matriz inversa por la matriz homogénea de transformación del segundo marcador con respecto al sistema de referencia de la cámara.
   Esto fue posible utilizando los siguientes comandos:

```
vpHomogeneousMatrix invMat = cMo_vec[0].inverse();
vpPoint p1(0.0, 0.0, 0.0);
p1.changeFrame(invMat * cMo_vec[1]);
```

4. Haciendo la multiplicación de los comandos anteriores se obtienen las coordenadas en tres dimensiones de los marcadores con respecto al origen global. Estas coordenadas son las que se envían por protocolo UDP a una dirección ip específica para poder hacer uso de dicha información desde otra aplicación. Para esto se realizó lo siguiente:

```
char* direccion = "127.0.0.1";
int puerto = 666;

WSADATA wsaData;
if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
	fprintf(stderr, "Could not open Windows connection.\n");
}

SOCKET sd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (sd == INVALID_SOCKET) {
	fprintf(stderr, "Could not create socket.\n");
	WSACleanup();
}

struct sockaddr_in server;
```

```
server.sin_family = AF_INET;
server.sin_port = htons(puerto);
server.sin_addr.s_addr = inet_addr(direccion);

char msgch[35] = "M";
strcat(msgch, xstring.c_str());
strcat(msgch, ystring.c_str());
strcat(msgch, zstring.c_str());
strcat(msgch, tag_id1_s.str().c_str());
strcat(msgch,"M");
strcat(msgch, xstring.c_str());
strcat(msgch, ystring.c_str());
strcat(msgch, zstring.c_str());
strcat(msgch, tag_id1_s.str().c_str());

sendto(sd, msgch, strlen(msgch), 0, (SOCKADDR *)&server, sizeof(server));
```

5. También es posible obtener el número del marcador para poder hacer uso de los datos de diferente manera en función del marcador que está proporcionando la información. Esto se hace mediante los siguientes comandos:

```
std::string message1 = detector.getMessage(1);
std::size_t tag_id_pos1 = message1.find("id: ");
int tag_id1;
std::stringstream tag_id1_s;
if (tag_id_pos1 != std::string::npos) {
  tag_id1 = atoi(message1.substr(tag_id_pos1 + 4).c_str());
  tag_id1_s << tag_id1;
}
```

Una vez que se ha realizado la programación correspondiente, y tras compilar el programa (por ejemplo haciendo uso de Visual Studio), se puede utilizar el programa corriendo el ejecutable que se localiza en la carpeta: C:\visp-ws\visp-build-vc16\tutorial\detection\tag\Release.

Con esto ya es posible extraer la información de la posición y orientación en tres dimensiones para cada marcador, así como enviar dicha información mediante el protocolo UDP para su uso en otros programas, como por ejemplo en Matlab.

Recepción de datos mediante protocolo UDP desde Simulink

A continuación, se explica la manera en que pueden recibirse datos que hayan sido enviados mediante UDP desde un programa en Simulink, sin embargo, este procedimiento podía adaptarse para hacer uso de la información proporcionada por AprilTag desde algún otro tipo de programa.

Como primer paso se debe realizar la instalación de Simulink Desktop Real-Time, ya que esto es necesario para poder interactuar con dispositivos en tiempo real, en este caso se utilizará para realizar la comunicación mediante el protocolo UDP.

Para instalarlo se debe introducir en la ventana de comandos de Matlab la siguiente instrucción:

```
sldrtkernel -install
```

En seguida se deberá confirmar para continuar con la instalación, en caso de tener ya instalado esto o tener una versión diferente se mostrará un mensaje indicando la situación.

Una vez finalizada la instalación se deberá verificar que la instalación fue completada de manera correcta mediante el comando:

```
rtwho
```

Con esto se deberá mostrar la versión de Simulink Desktop Real-Time que se tiene instalada. Para más información sobre este *kernel* se puede hacer referencia a la documentación de Matlab en el siguiente enlace:

https://la.mathworks.com/help/sldrt/ug/real-time-windows-target-kernel.html

Una vez finalizada la instalación de Simulink Desktop Real-Time se procederá a crear un modelo nuevo de Simulink, este se puede iniciar introduciendo el comando ***simulink*** en la ventana de comandos de Matlab y seleccionando la opción para crear un nuevo modelo en blanco.

Con la instalación que se realizó de Simulink Desktop Real-Time se incluyen también los bloques necesarios para recibir y enviar datos mediante el mismo protocolo UDP (Figura 4). Para recibir los datos en Simulink se utilizará el bloque "Packet Input", el cual se encuentra en la librería de bloques de Simulink en la sección llamada "Simulink Desktop Real-Time". Así mismo, en la misma sección también se encuentra el bloque que puede ser utilizado para enviar datos mediante este protocolo, "Packet Output".

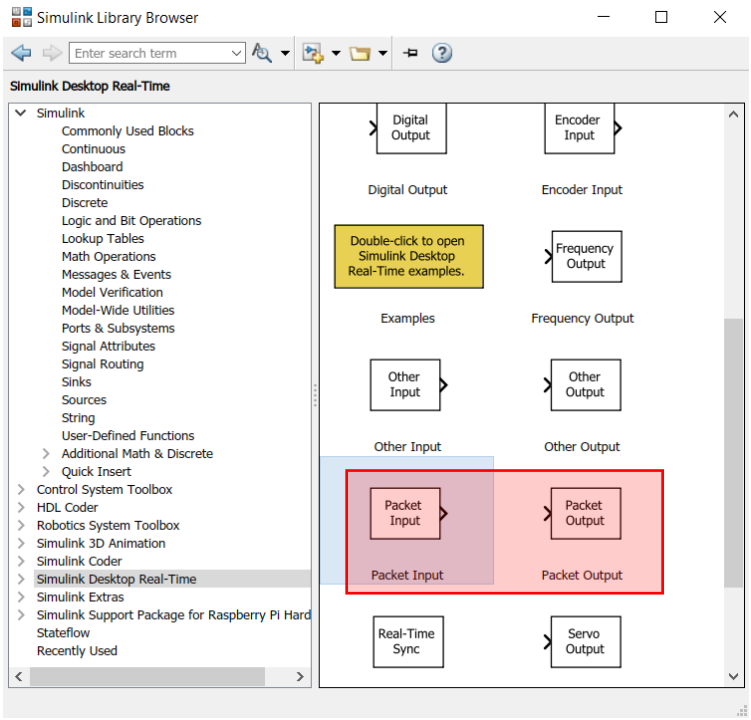


Figura 4. Bloques para el envío y recepción de datos en Simulink.

Ya que nuestro objetivo es recibir los datos que se envían desde el otro programa necesitamos utilizar el bloque “Packet Input”, éste puede arrastrarse con el ratón para colocarlo dentro del modelo en blanco de Simulink. Después de colocar el bloque, se debe configurar haciendo doble clic sobre él, lo cual deberá desplegar una ventana con opciones de configuración como la que se presenta en la figura 5.

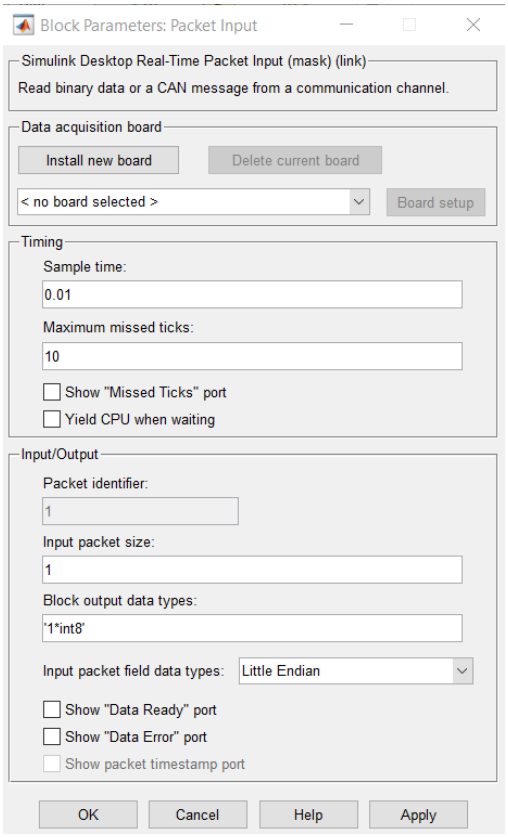


Figura 5. Configuración del bloque Packet Input en Simulink.

Aquí se observan varias opciones de configuración:

- Sample time: es el tiempo de muestreo que se considerará entre los paquetes.

- Maximum missed Ticks: es la cantidad de paquetes que pueden no recibirse antes de que se detecte como un error. Recordemos que el protocolo UDP realiza el envío de datos sin verificación, por lo que existe la posibilidad de perder paquetes de datos. Al configurar esta opción como “inf” se indica que el programa continúe sin importar la cantidad de paquetes de datos que no sean recibidos.
- Input packet size: indica la cantidad de datos que van a leerse en cada paquete, en este caso puede ser la cantidad de símbolos que contiene la cadena que se quiere recibir.
- Block output data types: indica los tipos de datos que se tendrán a la salida del bloque. Por ejemplo, en este caso suponemos que queremos recibir una cadena que tiene un número de un solo dígito y queremos que salga como entero, por lo tanto, se introduce *'1*int8'*. Si quisiéramos leer una cadena de 7 caracteres se puede expresar como *'1*int8'*. Con esto, se tendrá la cantidad de salidas especificadas, una por cada carácter de la cadena que se leyó.

  Importante: con esta configuración la salida será el número correspondiente al código ASCII del símbolo. Por ejemplo, si se recibe la cadena ‘1’, la salida del bloque será el número 49, ya que este valor es el que corresponde al símbolo ‘1’ de acuerdo con el código ASCII. Una forma de convertirlo al número correcto sería restando 48 del número que sale del bloque. Una tabla con el código ASCII puede encontrarse en la siguiente liga: https://elcodigoascii.com.ar/.

Para configurar el protocolo, la dirección y el puerto con el que se realizará la comunicación se debe dar clic en el botón “Install new board” (Figura 6), y a continuación, seleccionar la opción “Standard Devices” → ”UDP Protocol”, con lo que se abrirá una nueva ventana (figura 7). En esta nueva ventana se deberán introducir los datos correspondientes a la dirección IP y el puerto que se están utilizando para la comunicación, estos datos deben coincidir con los

que se están utilizando para el envío del paquete desde el programa de AprilTag.

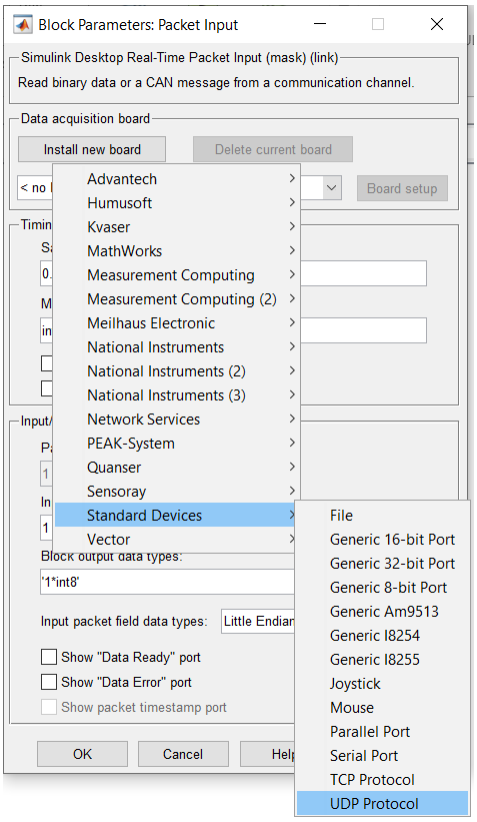


Figura 6. Nueva tarjeta para comunicación mediante el protocolo UDP.

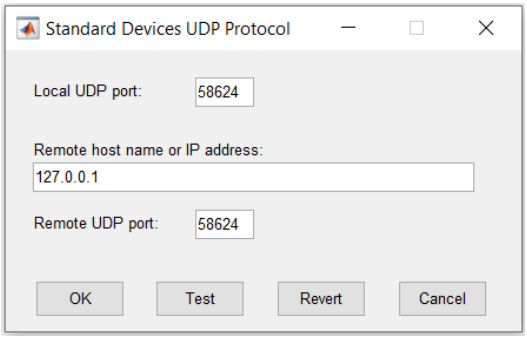


Figura 7. Configuración de la dirección IP y el puerto para la comunicación.

Tras haber introducido los datos es recomendable dar clic en el botón “Test”, con la finalidad de que el puerto y dirección IP estén disponibles para la conexión. De ser así se mostrará el mensaje de confirmación que se muestra en la Figura 8.

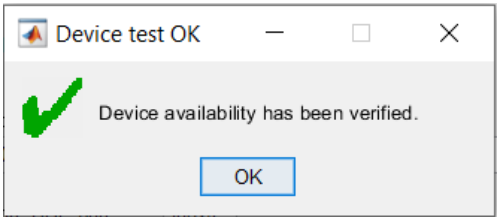


Figura 8. Verificación de la disponibilidad de la dirección IP y puerto.

Finalmente, se deben confirmar los cambios y guardar la tarjeta para la comunicación (Figura 9).

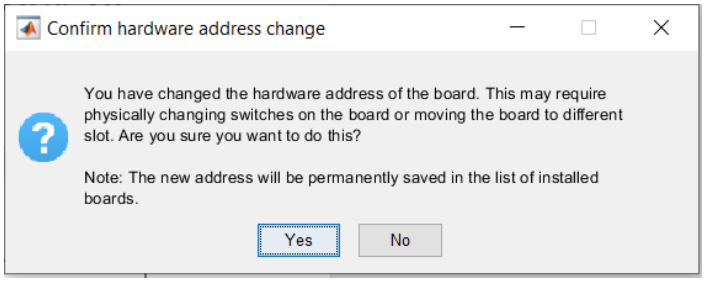


Figura 9. Confirmación de cambios y guardar la tarjeta de comunicación.

*Es importante recordar que tras hacer modificaciones en las configuraciones se debe seleccionar “Apply” o “OK” en la ventana correspondiente para que los cambios tengan efecto.

Regresando al modelo en donde se colocó el bloque, también será necesario crear un elemento que nos permita visualizar los datos que se están obteniendo. Una posibilidad es utilizando el elemento “Display”, ubicado dentro de la librería de bloques de Simulink en la sección “Simulink” → “Sinks” (Figura 10), el cual puede conectarse a la salida del bloque “Packet Input”.

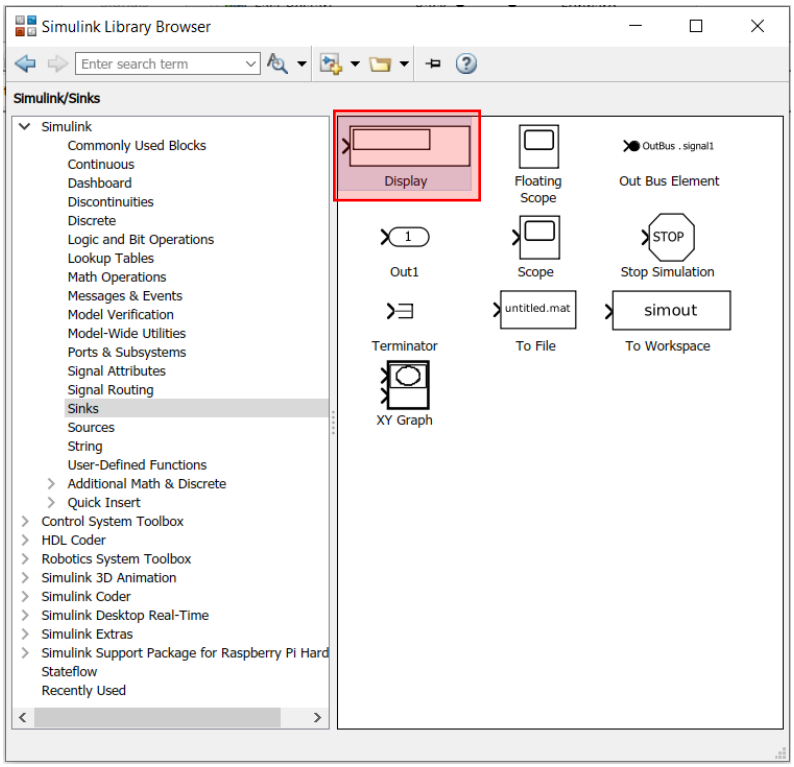


Figura 10. Elemento “Display” para visualizar datos.

Para probar la comunicación es necesario guardar el modelo y correrlo haciendo clic en el botón “Run” ubicado en la parte superior de la ventana del modelo de Simulink (Figura 11). También es necesario indicar el tiempo en segundos que durará la simulación en el recuadro “Stop Time” (figura 11), inicialmente viene definido como 10.0 lo que quiere decir que la simulación se detendrá después de 10 segundos. Para mantener la simulación corriendo indefinidamente se puede introducir como valor “inf”.

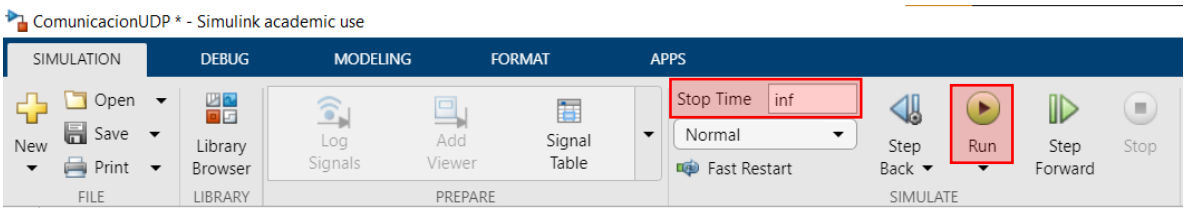


Figura 11. Botón para correr y especificar el tiempo de simulación.

Una vez corriendo el modelo de Simulink se deberá correr también el programa de AprilTag ya con la configuración para el envío de datos mediante el socket con protocolo UDP. Si todo se realizó de manera correcta se deberá visualizar en el Display en Simulink el dato enviado.

Para este ejemplo se envió como mensaje, desde el programa de AprilTag, la cadena "1". Como se observa en la figura 12, el dato leído desde Matlab es el entero 49, que corresponde al Código ASCII del número "1".

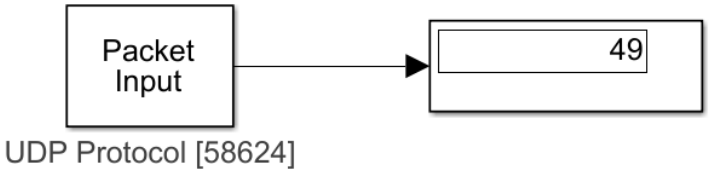


Figura 12. Recepción de datos en Simulink.

Con esto ya se tienen las herramientas básicas necesarias para enviar los datos desde AprilTag hasta Simulink y poder hacer uso de esta información en Simulink para desarrollar programas más complejos de control de robots móviles.

Realiza las siguientes actividades para continuar con la práctica:

- Prueba la comunicación mediante el protocolo UDP entre el programa de AprilTag y Simulink enviando un solo dígito como se mostró en el desarrollo de la práctica.
- Realiza las modificaciones necesarias en el programa de AprilTag para que se envíen como una cadena de texto los datos correspondientes al número del marcador identificado, las coordenadas y la orientación del marcador.
- Desarrolla en Simulink un programa que reciba esta cadena, separe la información (número de marcador, coordenada X, coordenada Y, ángulo) y la muestre de forma numérica.

Recomendaciones:

- Dado que el protocolo UDP no tiene verificación en la recepción de los paquetes se recomienda enviar la cadena con un símbolo adicional que permita identificar en dónde inicia una cadena y dónde inicia la siguiente. Además, para facilitar la separación de los datos se puede mandar la misma cantidad de dígitos para cada dato, completando con ceros donde haga falta.
  Por ejemplo, si se quiere mandar id=1, x=127, y=34 y ang=65, utilizando como identificador de inicio de cadena el carácter “M”, la cadena podría quedar de la siguiente manera:
  “**M**1**127**034**065**”
- Para el programa de Simulink se recomienda utilizar un bloque “MATLAB Function” (figura 13), en donde se puede programar lo necesario para identificar el inicio de la cadena con el identificador, la separación de la cadena en los dígitos que corresponden a cada dato y la conversión a valores numéricos utilizando el código ASCII.
  Más información sobre “MATLAB function”:
  https://la.mathworks.com/help/simulink/slref/matlabfunction.html

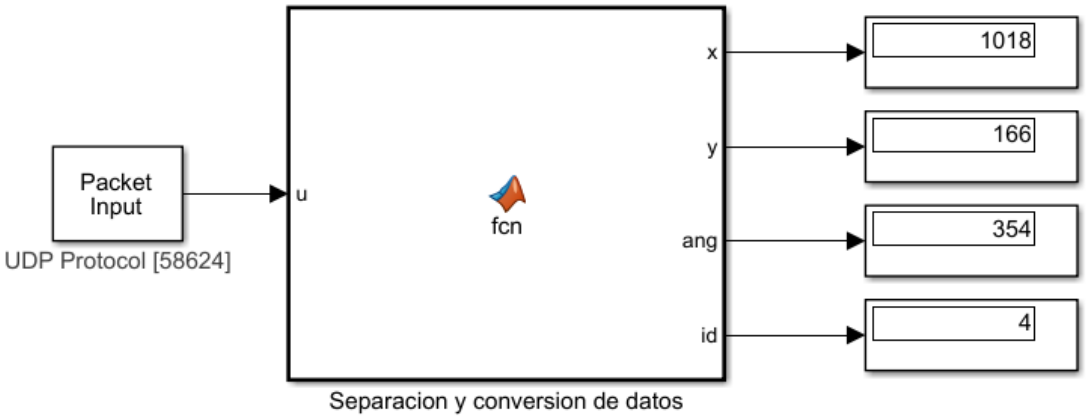


Figura 13. Propuesta de implementación en Simulink.

c) Implementación de AprilTag directamente con Matlab

En la guía de usuario de AprilTag que se encuentra en el repositorio (https://github.com/AprilRobotics/apriltag/wiki/AprilTag-User-Guide), se indica que ya han sido desarrollados por terceros programas para el soporte de AprilTag desde Matlab (https://github.com/alddiaz/MATLAB_AprilTag3). Esta opción no ha sido revisada a detalle, sin embargo, podría ser de utilidad para facilitar la integración de AprilTag con Matlab.

8. Resultados

Como resultados de esta práctica se deben entregar los programas realizados para lograr la obtención de la posición y orientación de los marcadores, así como un video en el que se explica y se demuestra el funcionamiento.

9. Conclusiones

La obtención de la posición y orientación de un robot es parte fundamental para lograr el comportamiento deseado en robótica móvil. El uso de cámaras y sistemas de visión con marcadores brindan la posibilidad de medir esto bajo ciertas condiciones, sin la necesidad de utilizar sensores más complejos o caros.

En esta práctica se realizó una introducción al software de AprilTag, con el cual es posible identificar marcadores dentro del área de visión de una cámara y obtener la información relativa a su posición y orientación. Esta información será de gran utilidad para poder programar y controlar robots móviles en prácticas subsecuentes. Así mismo, se introdujo la comunicación mediante el protocolo UDP que es una buena opción para enviar los datos hacia otro tipo de programas que facilitan la programación en el ámbito de la robótica móvil como es el caso de Matlab/Simulink.

Bibliografía

- Olson, E. “AprilTag: A robust and flexible visual fiducial system.” *Proc. IEEE ICRA.* May 2011.
- Wang, J. and E. Olson. “AprilTag 2: Efficient and robust fiducial detection.” *Proc. IEEE/RSJ IROS.* Oct 2016.
- Krogius, M., Haggenmiller, A. and E. Olson. “Flexible Layouts for Fiducial Tags.” *Proc. IEEE/RSJ IROS*. Oct 2019.