IPD482 Guia 2

Patricio Carrasco O'Ryan
Ingeniería Civil Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
patricio.carrascoo@sansano.usm.cl

Sebastian Espinoza Toro
Ingeniería Civil Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
sebastian.espinozat@sansano.usm.cl

Tomás Bernal
Ingeniería Civil Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
tomas.bernal@sansano.usm.cl

***Abstract*—Este informe se centra en la estimación de la orientación de un trailer utilizando la información proporcionada por un LiDAR. Se proponen y explican dos técnicas diferentes para lograr este objetivo, incluyendo las ecuaciones necesarias y pruebas de consistencia con el ground-truth. Además, se realiza una estimación de la carga computacional de cada técnica. El trailer en cuestión es un "full-trailer", que se considera como dos trailers tipo uniciclo que comparten un mismo vagón. Se utilizó un LiDAR 2D montado sobre el robot que está apuntado a la cara frontal del trailer. Las lecturas de los puntos que rebotan con el cuerpo del trailer se utilizan para determinar el ángulo del eje posterior (trailer 2). Se colocó un cilindro metálico sobre la barra de conexión entre el robot y el trailer con el objetivo de ocluir el haz y usar esa información para determinar el ángulo del eje delantero (trailer 1). El informe incluye un análisis detallado de cada técnica y una comparación basada en métricas. El código desarrollado se encuentra disponible en https://github.com/sebastianespinozat/ipd482-t2.**

***Index Terms*—Robots móviles, LIDAR, Full-trailer, Estimación , Uniciclo.**

I. Método 1

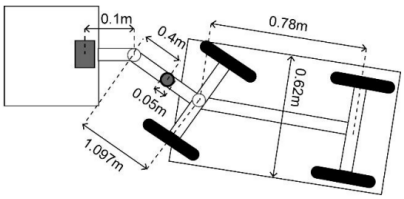


Fig. 1: Geometría desde el LIDAR hasta el Full-Dolly y sus respectivos Join

Para el primer método se comienza considerando la posición del cilindro en el espacio. Trabajamos en coordenadas polares y seguimos la geometría desde el LIDAR hasta el primer punto de unión, que está fijado a 0.1m. El cilindro tiene libertad de giro con un radio de 0.4m, como se muestra en la Figura 1.

En el peor de los casos, el ángulo entre el primer punto de unión y la posición del cilindro es de $\pm\frac{\pi}{2}$. Aplicando el teorema de Pitágoras, determinamos el valor de la hipotenusa, que representa la distancia entre el LIDAR y la posición del cilindro. En este caso, la hipotenusa representa la distancia mínima a la que puede encontrarse el cilindro, que es de 0.4123m.

Por otro lado, cuando el ángulo entre el primer punto de unión y el cilindro es de π, se representa la mayor distancia a la que puede encontrarse el cilindro desde la perspectiva del LIDAR. Esta distancia es la suma de las dos medidas desde el LIDAR hasta el punto de unión, más la distancia desde el punto de unión hasta la posición más alejada del cilindro, considerando su diámetro. Por lo tanto, la distancia máxima a la que se considera que puede encontrarse el cilindro es de 0.55m.

Al trabajar con los datos del LIDAR en coordenadas polares, establecemos una condición de búsqueda en los índices de cada instante de medición del sensor que cumpla con estas distancias mínima y máxima de 0.4123m y 0.55m, respectivamente, entre los ángulos 90° y -90° del recorrido que realiza el LIDAR, que corresponden a los máximos ángulos que puede alcanzar el cilindro en el peor caso propuesto . Estos dos radios se grafican en coordenadas polares, como se muestra en la Figura 2.

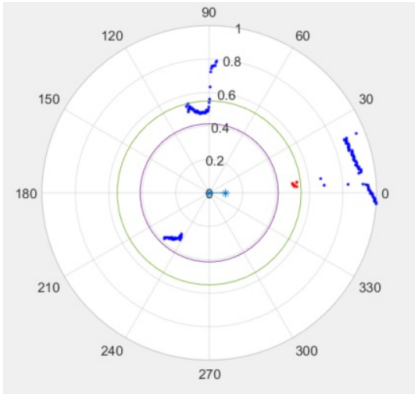


Fig. 2: Área entre Radio mínimo y Radio máximo donde se puede en contar el cilindro

Con este primer filtro de puntos, inferimos que el cilindro debe ubicarse en alguno de los puntos observados. Para determinar cuál de los puntos filtrados de la nube pertenece al cilindro, empleamos el Error Cuadrático Medio (ECM) en el vector de ángulos entre todos los puntos que cumplen la condición previa. Esto nos permite evaluar la dispersión de los puntos a lo largo del recorrido propuesto.

Inicialmente, establecemos un umbral de error de 0.001. Si todos los puntos capturados tienen un ECM menor a este umbral, interpretamos que todos los puntos filtrados están muy próximos entre sí. Por lo tanto, consideramos directamente que estos puntos forman parte del cilindro que buscamos detectar.

Por otro lado, si el ECM supera el umbral establecido, concluimos que existen más puntos dispersos que cumplen la condición de filtrado. En este caso, tomamos todo el espectro de ángulos detectados en ese instante y lo dividimos en dos, generando así dos nuevas áreas para analizar y detectar el cilindro.

Al tener dos áreas, recalculamos el ECM para cada una de ellas de manera independiente. La zona que presente un error menor al umbral de 0.001 se considerará como la ubicación del cilindro. Si ninguna zona cumple con el umbral, seleccionamos la zona con el ECM más bajo y repetimos el paso anterior, dividiéndola en dos y repitiendo el proceso hasta encontrar una zona donde el ECM cumpla con el umbral propuesto. También se compara la detección del cilindro actual con la medición del cilindro de la iteración anterior, con el objetivo de mantener la detección anterior en caso de que la nueva posición del cilindro haya cambiado de forma drástica producto de alguna detección errónea o si simplemente no hay puntos detectados producto de algún tercero irrumpiendo la medición del LIDAR.

Una vez identificados los puntos que representan al cilindro, convertimos la media del ángulo de los puntos del cilindro y el radio mínimo del punto más cercano al LIDAR de coordenadas polares a cartesianas. Estos puntos en coordenadas cartesianas, denominados xR e yR, representan la posición del cilindro. Con estos puntos, trazamos dos vectores. El primero, denominado v1, representa el vector en el eje X desde el origen hasta 1 metro en el mismo eje. El segundo vector, v2, se traza desde la primera unión (0.1m) hasta las coordenadas cartesianas obtenidas del cilindro. Como se puede observar en el código de Matlab 1, calculamos el ángulo entre estos dos vectores. Este cálculo siempre arrojará un resultado positivo, por lo que el signo del ángulo será el mismo que el valor de la coordenada yR del cilindro. Este ángulo corresponde a θ_1.

Listing 1: θ_1 mediante ángulo de vectores

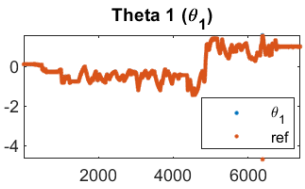
```
[xR,yR]=pol2cart(MeanAng, min(Cylinder));
v1 = [1,0,0] - [0,0,0];
v2 = [xR(1),yR(1),0] - [0.1,0,0];

if yR(1) > 0
B1=atan2(norm(cross(v1,v2)),dot(v1,v2));
else
B1=-atan2(norm(cross(v1,v2)),dot(v1,v2));
end
```

Una vez calculado θ_1 siguiendo la geometría nuevamente de la figura 1 se calcula mediante trigonometría los valores de x_1 e y_1 mediante las siguientes ecuaciones :

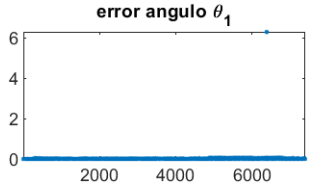
$$x_1 = 0.1 + 1.097cos(\theta_1) \tag{1}$$

$$y_1 = 1.097sin(\theta_1) \tag{2}$$

En las figuras 3, 4 y 5, se presentan los cálculos de θ_1, X_1 e Y_1 respectivamente, utilizando el algoritmo propuesto hasta este punto. Además, se muestran los valores de referencia (ground-truth) para la validación del algoritmo y se traza una medida del error absoluto entre el cálculo y el valor real de la coordenada.

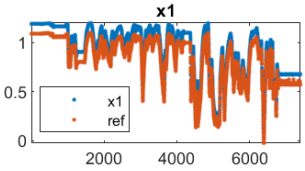


(a) Calculado de θ_1 por el método 1 junto a su ground-truth .

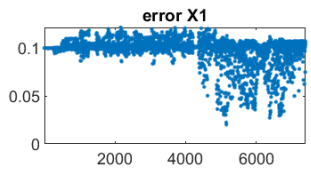


(b) Error entre referencia y cálculos de θ_1 con el método 1 .

Fig. 3: θ_1 obtenido mediante el método 1

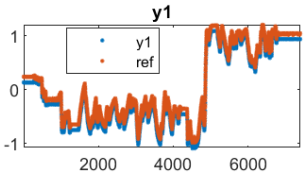


(a) Calculado de x_1 por el método 1 junto a su ground-truth .

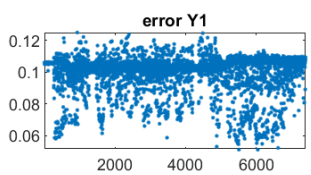


(b) Error absoluto entre referencia y cálculos de x_1 con el método 1 .

Fig. 4: x_1 obtenido mediante el método 1



(a) Calculado de y_1 por el método 1 junto a su ground-truth .



(b) Error entre referencia y cálculos de y_1 con el método 1 .

Fig. 5: y_1 obtenido mediante el método 1

El análisis de las gráficas revela que el valor de θ_1 calculado con el método 1 sigue la referencia de validación,

excepto cerca de la iteración 6200. Esta discrepancia se refleja en el error absoluto mostrado entre las dos mediciones de θ_1 y su valor de referencia, pero no es debido al método implementado, es por un punto que difiere con creces en la referencia. El Error Cuadrático Medio (MSE) es de 0.0428 para θ_1.

Al observar X_1 en la figura 4, se aprecia que los datos calculados siguen la referencia pero con un desfase (offset) de 0.1m. Este desfase se debe a que hay una unión a 0.1m de distancia del LIDAR en el eje x, que es fija y la referencia no está sumando esa distancia en sus cálculos. El error absoluto entre la referencia y el X_1 calculado se observa en la figura 4b y el MSE obtenido para este caso es de 0.0101.

Respecto a Y_1, la figura 5 muestra que los puntos calculados para Y_1 siguen su referencia, pero nuevamente aparece un desfase de 0.1m, para este caso no se encuentra una razón coherente para este desfase en la referencia de Y_1. Al igual que en el eje X, el error absoluto se observa en la figura 5b y el MSE de Y_1 es 0.0104.

Para el calculo de θ_2 , X_2 e Y_2, se parte con el supuesto de que en primera instancia se comienza viendo la cara frontal del trailer (condición inicial). El valor de θ_2 se obtiene como en ángulo formado por la pendiente perpendicular a la recta formada por la cara frontal y el eje horizontal, lo que es equivalente al ángulo formado entre el eje horizontal y la recta formada por la cara lateral del trailer. Esa similitud de los ángulos se muestra en la figura 6.

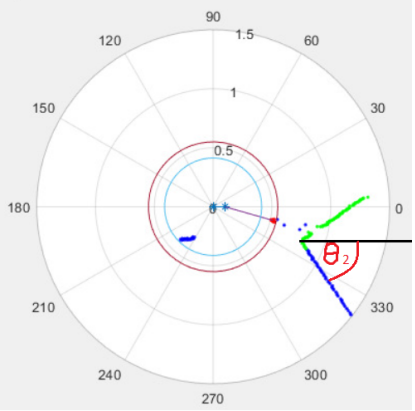


Fig. 6: Representación de θ_2 en el espacio polar

Una vez detectada la posición del cilindro en cada instante, se precisa de un filtrado adicional para eliminar puntos superfluos en los cálculos de detección del Full-trailer. Este proceso se lleva a cabo utilizando la información de la nube de puntos que corresponden al cilindro. Para ello, se calcula el promedio de los ángulos de cada punto dentro de este cilindro.

Contando con esa información y los datos de puntos en coordenadas polares, se procede a filtrar aquellos puntos que se encuentran a una distancia superior a 0.55m e inferior a 1.5m. Este rango se ha establecido teniendo en cuenta que es allí donde se presenta la mayor probabilidad de localizar al Full-trailer.

Finalmente, tras haber identificado todos los puntos que cumplen con esta condición, sólo se utilizan para cálculos aquellos que se encuentran a $\pm\frac{\pi}{6}rad$ del promedio del ángulo de los puntos del cilindro, y así se genera una ventana de búsqueda más acotada con respecto a la posición del cilindro.

Una vez obtenidos los datos en los que se puede encontrar el Full-trailer, se quiere detectar los puntos que pertenecen a la cara frontal o lateral que se perciben como lineas rectas. En el código MATLAB para la detección de rectas en datos 2D del LIDAR y, por ende, para el cálculo de ángulo θ_2, encontramos la implementación del algoritmo RANSAC. Este algoritmo se caracteriza por su robustez en la detección de modelos subyacentes en datos que pueden estar considerablemente contaminados por ruido y valores atípicos.

El código de la función de MATLAB proporcionada, titulada `aproximacion`, implementa este algoritmo. Esta función toma como entrada un conjunto de 'datos' (puntos en 2D del LIDAR) y un valor *minInliers* que define el mínimo número de puntos (inliers) requeridos para que una línea se considere óptima.

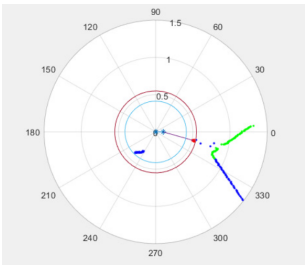
El algoritmo se ejecuta a lo largo de un número predeterminado de iteraciones, definido por `numIter`. Durante cada iteración, selecciona dos puntos al azar del conjunto de datos y calcula la línea que pasa por ellos. Posteriormente, mide la distancia de todos los puntos a esta línea y selecciona aquellos que se encuentran dentro de un umbral de distancia predefinido ('distThresh') como inliers.

Si el número de inliers es superior a 'minInliers' y al número máximo de inliers hasta el momento, se actualiza el "mejor modelo", es decir, la línea que mejor se ajusta a la mayoría de los puntos del conjunto de datos. Además, si el error entre el modelo actual y el anterior es menor que un valor umbral ('error'), el algoritmo se detiene prematuramente, considerando que ha encontrado el modelo óptimo.

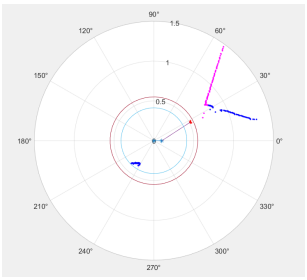
Sin embargo, para garantizar que no se queda sin un modelo aceptable en caso de que ninguna de las líneas calculadas cumpla con el criterio de 'minInliers', el algoritmo también mantiene un registro del "peor modelo". Este es el modelo que, aunque no cumple con 'minInliers', ha obtenido el mayor número de inliers en todas las iteraciones.

Una vez que se ha completado el algoritmo, se devuelven los parámetros de la línea del "mejor modelo" (o del "peor modelo", si ningún modelo ha alcanzado 'minInliers'). Estos parámetros corresponden a la pendiente ('m') y la intersección ('b') de la línea, y se utilizan posteriormente para calcular el ángulo θ_2 y llevar a cabo la detección del Full-trailer. En la Figura 7 se puede observar la forma en que detecta la cara frontal o lateral del Dolly en coordenadas polares, los datos y cálculos realizados por este algoritmo esta en cartesianos.

Bajo la condición inicial propuesta, en la que el LIDAR detecta únicamente la cara frontal del Dolly, este supuesto nos permite discernir qué cara está siendo detectada en cada momento mediante el algoritmo RANSAC. La clave para esto radica en la pendiente que identifica el algoritmo, la cual está asociada a un estado específico del Dolly. Estos estados se denominan 'lat' (lateral) y 'front' (frontal).



(a) Detección de la Cara frontal .



(b) Detección de la cara Lateral

Fig. 7: Detección de rectas con RANSAC

Al almacenar la pendiente precedente y compararla con la pendiente actual, si se observa una variación aproximada de 90° entre ambas, se considera que el estado del Dolly debe cambiar en función del estado actual. La implementación de esta lógica se ilustra a continuación en formato de código.

Listing 2: Comparación de pendientes

```
if m_a*m > -2.5 && m_a*m < -0.1 && i ~= 1
        if contains(stateDolly, 'front')
            stateDolly = 'lat';
        else
            stateDolly = 'front';
        end
    end
%   m_a : pendiente anterior
```

Según el estado del Dolly es como se calcula el ángulo θ_2 mediante el siguiente código en Matlab :

Listing 3: Calculo θ_2

```
% Calculo Beta2
if contains(stateDolly, 'front')
    B2(i,1) = atan(-1/m);
elseif contains(stateDolly, 'lat')
    B2(i,1) = atan(m);
end

if i ~= 1 && (B2(i,1) - B2(i-1,1) > pi/2)
    B2(i,1) = -pi + B2(i,1);
elseif i ~= 1 &&(B2(i,1)-B2(i-1,1)< -pi/2)
    B2(i,1) = pi + B2(i,1);
end
```

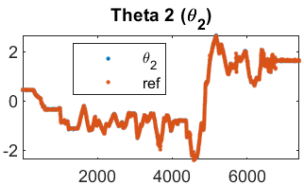
El segundo bloque 'if' del código 3 corrige cambios muy grandes en los ángulos con respecto a su anterior debido a los cuadrantes en los que se encuentra , ya que el arco tangente por si solos en los casos que se describe un cambio muy grande de ángulo, hay que sumar π o restarle π según sea el caso.

Una vez calculado θ_2 siguiendo la geometría nuevamente de la figura 1 se calcula mediante trigonometría los valores de x_2 e y_2 mediante las siguientes ecuaciones :

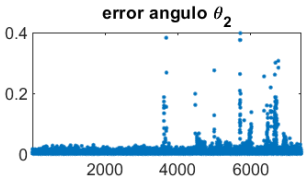
$$x_2 = x_1 + 0.78cos(\theta_2) \tag{3}$$

$$y_2 = y_2 + 0.78sin(\theta_2) \tag{4}$$

En las Figuras 8 , 9 y 10 se puede observar el cálculo de θ_2 , X_2 e Y_2 respectivamente del algoritmo planteado, junto con el ground-truth para validar el algoritmo además de una medida de error absoluto entre el calculo y el valor real de la coordenada también graficada.

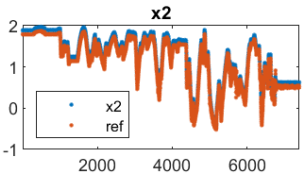


(a) Calculado de θ_2 por el método 1 junto a su ground-truth .

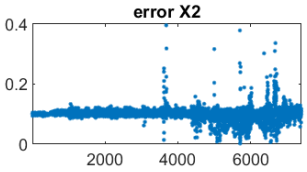


(b) Error entre referencia y cálculos de θ_2 con el método 1 .

Fig. 8: θ_2 obtenido mediante el método 1

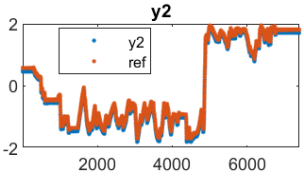


(a) Calculado de x_2 por el método 1 junto a su ground-truth .

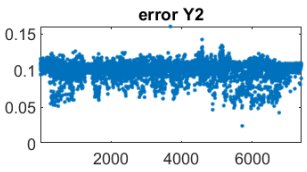


(b) Error entre referencia y cálculos de x_2 con el método 1 .

Fig. 9: x_2 obtenido mediante el método 1



(a) Calculado de y_2 por el método 1 junto a su ground-truth .



(b) Error entre referencia y cálculos de y_2 con el método 1 .

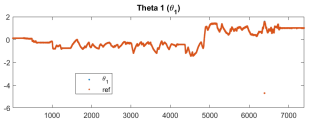
Fig. 10: y_2 obtenido mediante el método 1

Como el cálculo de X_1 e Y_1 tienen un offset de 0.1m en sus cálculos, ese error también se ve arrastrado en X_2 e Y_2 por la forma en que se calculan estos, que dependen del valor de X_1 e Y_1 respectivamente. Para θ_2 su valor de MSE es de 0.0026 , para X_2 su valor de MSE es de 0.0108 y finalmente el MSE de Y_2 es de 0.0111 .

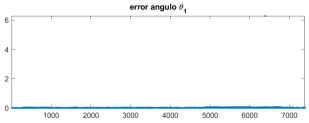
II. MÉTODO 2

Para implementar este método, se modifica la manera de calcular θ_2. La detección del cilindro en la nube de puntos del LIDAR se mantiene igual que en el método 1. Del mismo modo, los cálculos empleados para obtener X_1, Y_1 y θ_1 no presentan cambios, es decir, son los mismos que se utilizan en el método 1.

Como se puede observar en las figuras 11, 12 y 13, estas siguen las referencias del ground-truth y se presentan de la misma forma que en el método 1. Cabe destacar que para X_1 y Y_1, existe un offset de 0.1m, de igual forma que en el método 1.

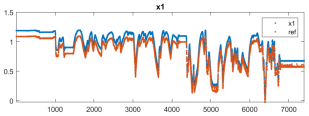


(a) Calculado de θ_1 por el método 2 junto a su ground-truth .

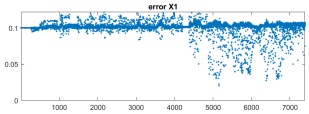


(b) Error entre referencia y cálculos de θ_1 con el método 2 .

Fig. 11: θ_1 obtenido mediante el método 2

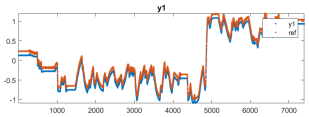


(a) Calculado de x_1 por el método 2 junto a su ground-truth .

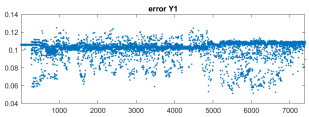


(b) Error entre referencia y cálculos de x_1 con el método 2 .

Fig. 12: x_1 obtenido mediante el método 2



(a) Calculado de y_1 por el método 2 junto a su ground-truth .



(b) Error entre referencia y cálculos de y_1 con el método 2 .

Fig. 13: y_1 obtenido mediante el método 2

Para obtener y saber la orientación del Dolly aplicamos la siguiente metodología :

1) Se realiza un filtrado angular de los datos basándose en la media de los ángulos del cilindro, utilizando un margen de tolerancia de $\pm\frac{\pi}{6}$ radianes. De esta manera, se seleccionan aquellos puntos que están dentro de este rango angular respecto al cilindro.
2) Luego, para detectar las extremidades del dolly, se buscan puntos que se encuentran en ciertos rangos alrededor de los ángulos extremos del cilindro. Para detectar la parte superior del dolly, se buscan puntos que se encuentran entre 0.8 y 5 grados por encima del ángulo máximo del cilindro. Similarmente, para detectar la parte inferior del dolly, se buscan puntos que se encuentran entre 0.8 y 5 grados por debajo del ángulo mínimo del cilindro. Este filtrado angular adicional ayuda a detectar pequeñas porciones de la parte superior e inferior del dolly (parte superior e inferior que se genera cuando el cilindro obstaculiza el LIDAR).
3) Posteriormente, se analizan las medias de los radios de los puntos seleccionados en los pasos anteriores para determinar la orientación del Dolly. Según las condiciones establecidas:
   - Si la media de los radios tanto superior como inferior es mayor que 0.85m, se considera que Dolly tiene una orientación ”Frontal”.
   - Si la media de los radios tanto superior como inferior es menor que 0.85m, se verifica si el Dolly se encuentra en una ”esquina Derecha” o en una ”esquina Izquierda”, para reorientarlo hacia una posición ”Lateral Derecha” o ”Lateral Izquierda”, respectivamente.
   - Si la media del radio superior es menor que 0.85m y la del radio inferior es mayor que 0.85m, se considera que Dolly está en la ”esquina Derecha”.
   - Si la media del radio superior es mayor que 0.85m y la del radio inferior es menor que 0.85m, se considera que Dolly está en la ”esquina Izquierda”.

De igual forma que en el método 1, se desea partir de un caso favorable para facilitar los cálculos de θ_2 más adelante, este caso inicial es con el LIDAR mirando la cara frontal del Dolly. A continuación en la figura 14 se muestran los estados y orientación del dolly que puede detectar este algoritmo.

Después de detectar el estado del dolly , se procede a calcular la pendiente con los puntos detectados, la función está implementada en Matlab y se llama ’pendienteDolly’.

Los argumentos de entrada para esta función son:

- ’angSupDolly’ y ’radSupDolly’ : El ángulo y el radio de los puntos superior del dolly.
- ’angInfDolly’ y ’radInfDolly’ : El ángulo y el radio de los puntos inferior del dolly.
- ’stateDolly’: El estado del dolly, que puede ser ’Frontal’, ’Lateral Izquierda’, ’Lateral Derecha’, ’esquina Izquierda’, ’esquina Derecha’, u otros.

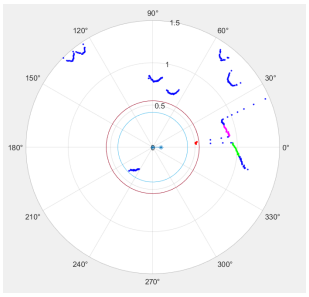
Inicialmente, la función convierte las coordenadas polares del punto superior e inferior a coordenadas cartesianas usando la función ’pol2cart’.

Después, la función chequea el estado del dolly y calcula la pendiente de manera correspondiente. Las condiciones son las siguientes:

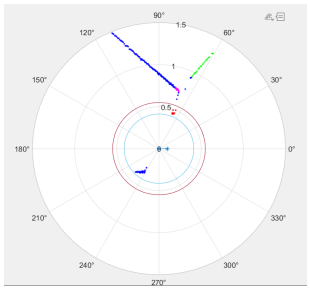
- Si el estado del dolly es ’Frontal’, la pendiente se calcula utilizando tanto el punto superior como el punto inferior del dolly. Se usa la función ’polyfit’ para hacer un ajuste lineal de los datos y luego se toma la inversa negativa de la pendiente calculada.
- Si el estado del dolly es ’Lateral Izquierda’ o ’Lateral Derecha’, la pendiente se calcula de la misma manera

que en el caso 'Frontal', pero aquí no se toma la inversa negativa.

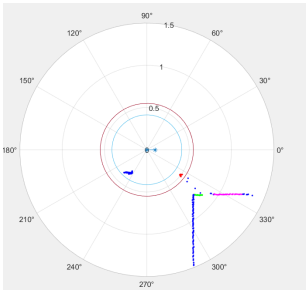
- Si el estado del dolly es 'esquina Izquierda', la pendiente se calcula solo utilizando los puntos superiores del dolly. Se usa la función polyfit para hacer un ajuste lineal y luego se toma la inversa negativa de la pendiente calculada.
- Si el estado del dolly es 'esquina Derecha', la pendiente se calcula solo utilizando los puntos inferiores del dolly, de manera similar al caso 'esquina Izquierda'.



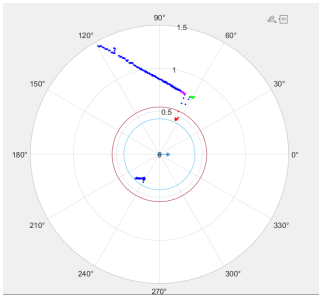
(a) detecta cara frontal



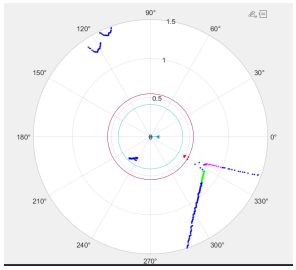
(b) detecta esquina derecha



(c) detecta esquina izquierda



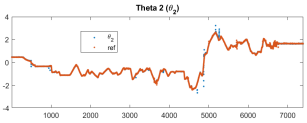
(d) detecta cara lateral derecha



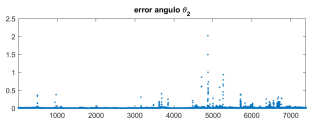
(e) detecta cara lateral izquierda

Fig. 14: Orientaciones que puede detectar el método 2

Una vez obtenida la pendiente necearía para detectar θ_2 se aplican las misma formulas que en el método 1, que son (3) para obtener ese ángulo según la pendiente que se le da de entrada. Para X_2 e Y_2 se usan las ecuaciones 3 y 4 respectivamente. El MSE de θ_2 es de 0.0026 mientras que el de X_2 es 0.0108 y el de Y_2 es de 0.0111 . Sus figuras se muestran a continuación.

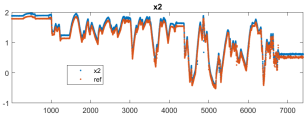


(a) Calculado de θ_2 por el método 2 junto a su ground-truth .

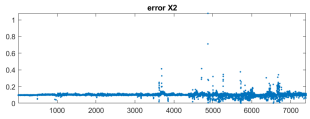


(b) Error entre referencia y cálculos de θ_2 con el método 2 .

Fig. 15: θ_2 obtenido mediante el método 2



(a) Calculado de x_2 por el método 2 junto a su ground-truth .



(b) Error entre referencia y cálculos de x_2 con el método 2 .

Fig. 16: x_2 obtenido mediante el método 2

III. Discusión

A. *Comparación entre métodos*

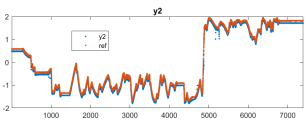
Para la detección del cilindro se utiliza el mismo algoritmo de detección, debido a su gran robustez y simplicidad. Esto genera que los resultados en las mediciones de θ_1, x_1 y y_1 sea idénticos para ambos métodos. Por lo tanto, los algoritmos utilizados para la estimación de θ_2, x_2 e y_2 son distintos. En la Tabla I se puede apreciar los errores cuadráticos medios (Mean squared error) de cada método respecto a las referencias entregadas.

MSE	Método 1	Método 2
θ_1	0.0428	0.0428
x_1	0.0101	0.0101
y_1	0.0104	0.0104
θ_2	$4.8116e-04$	0.0026
x_2	0.0105	0.0108
y_2	0.0104	0.0111

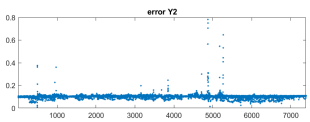
TABLE I: MSE para θ_1, x_1 e y_1 en cada método

Utilizando el MSE como métrica de evaluación, se tiene que el método 1 es ligeramente más preciso que el método 2 en las estimaciones de x_2 e y_2, además, de tener una clara mejoría respecto al método 2 en la estimación de θ_2.

Otra forma de evaluar el rendimiento de los métodos utilizados es ver el costo computacional que requieren. En la Tabla



(a) Calculado de y_2 por el método 2 junto a su ground-truth .



(b) Error entre referencia y cálculos de y_2 con el método 2 .

Fig. 17: y_2 obtenido mediante el método 2

II se puede apreciar un resumen en los tiempos de ejecución de cada método.

	Método 1	Método 2
$mean(t)$ s	0.002	0.0013
t_{max} s	0.3711	0.1667
t_{min} s	$7.273e-04$	$7.296e-04$
t_{total} s	14.6626	9.4513

TABLE II: Tiempos de ejecución obtenidos en cada método

Pese a que el método 1 es un mejor estimador de los valores, el método 2 es más rápido de ejecutar. Esta diferencia se debe a la aproximación de rectas para la detección de las caras del tráiler que se realiza en cada iteración del método 1, debiendo realizar hasta las N iteraciones definidas por cada muestra.

B. Limitaciones y desafíos

La principal limitación -y a su vez desafío- en el desarrollo de los métodos fue la detección de puntos que representaran las caras del tráiler, y a su vez determinar si correspondía a la cara frontal o lateral, pues la mitad de las estimaciones dependía de esa tarea. A su vez, se sumaba el problema de las manipulaciones de terceros con el robot.

IV. Conclusiones

Se puede concluir que existen una gran cantidad de formas de poder analizar y estimar las nubes de puntos de sensores LIDAR para su uso en la robótica, donde siempre existirá el trade off entre la precisión del método utilizado y el coste que este requiere. La estimación de ángulos y posiciones utilizando datos de LIDAR presenta distintos desafíos. La densidad de los puntos, el ruido de los mismos y las manipulaciones que genera el entorno sobre el robot utilizado afectan drásticamente los resultados, y obligan el desarrollo de algoritmos más robustos.