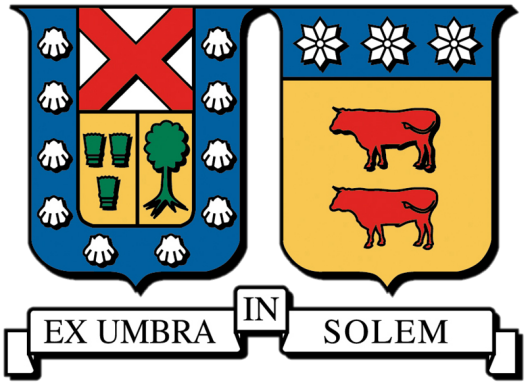
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO - CHILE



"ESTIMACIÓN DE ESTADOS DE VEHÍCULOS TERRESTRES USANDO ESTIMACIÓN DE HORIZONTE MÓVIL CON MEDICIONES REDUNDANTES DE GPS DE BAJO COSTE INCORPORANDO RESTRICCIONES GEOMÉTRICAS"

NICOLÁS IGNACIO MANSILLA MANSILLA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO

Guía: NESTOR NAHUEL DENIZ

Co-referente: FRANCISCO VARGAS

Marzo - 2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Estimación de estados de vehículos terrestres usando estimación de horizonte móvil con mediciones redundantes de GPS de bajo coste incorporando restricciones geométricas

Nombre del candidato(a): Nicolás Ignacio Mansilla Mansilla

Carrera / Grado: Ingeniería civil electrónica

Campus: Casa central **Departamento:** Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Nestor Nahuel Deniz, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

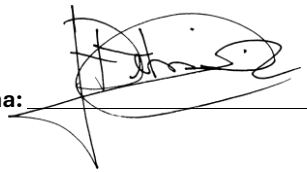
_____

_____

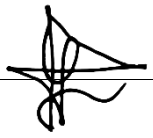
_____

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 03/11/2025 **Firma:** 

Estudiante o Candidato(a):

Fecha:03/11/2025 **Firma:** 

ESTIMACIÓN DE ESTADOS DE VEHÍCULOS TERRESTRES USANDO ESTIMACIÓN DE HORIZONTE MÓVIL CON MEDICIONES REDUNDANTES DE GPS DE BAJO COSTE INCORPORANDO RESTRICCIONES GEOMÉTRICAS

NICOLAS IGNACIO MANSILLA MANSILLA

Memoria para optar al título de Ingeniero Civil Electrónico, mención Control, submención Computadores.

Universidad Técnica Federico Santa María

Profesor Guía: Nestor Nahuel Deniz

Profesor Correferente: Francisco Vargas

Marzo - 2025

Resumen

En este proyecto, se implementa un algoritmo de Estimación de Horizonte Móvil (MHE, por las siglas en inglés de Moving Horizon Estimation) en Matlab para mejorar las mediciones proporcionadas por cuatro unidades GPS, dos de baja precisión y dos de alta precisión en sus mediciones, montados en un robot Husky A200. La estimación se compara directamente con un sensor de posición RTK (por sus siglas en inglés Real-Time Kinematic positioning) de precisión centimétrica y un filtro ampliamente utilizado en la industria y academia para la estimación de estados de sistemas dinámicos no lineales, el Filtro Extendido de Kalman (EKF, por las siglas en inglés de Extended Kalman Filter). MHE muestra ser un mejor algoritmo de estimación que EFK cuando se necesita seguir trayectorias que abarcan grandes distancias, especialmente cuando las condiciones iniciales son muy ruidosas, pero su rendimiento decae en entornos de área reducida y desplazamientos de curvas cerradas, llegando a ser similar al EKF. Aunque el rendimiento no cumplió con las expectativas establecidas por las simulaciones, se logra superar un método tradicional de estimación en entornos no ideales al aprovechar información a priori en forma de restricciones de optimización.

ESTIMATION OF GROUND VEHICLE STATES USING MOVING HORIZON ESTIMATION WITH REDUNDANT LOW-COST GPS MEASUREMENTS INCORPORATING GEOMETRIC CONSTRAINTS

NICOLAS IGNACIO MANSILLA MANSILLA

Thesis for the fulfillment of the B.S. in Electronic Engineering, major in Control, minor in Computer Electronics.

Universidad Técnica Federico Santa María

Advisor: Nestor Nahuel Deniz

Co-advisor: Francisco Vargas

March - 2025

Abstract

In this project, a Moving Horizon Estimation (MHE) algorithm is implemented in Matlab to improve the measurements provided by four GPS, two with low precision and two with high precision measurements, units mounted on a Husky A200 robot. The estimation is directly compared with a Real-Time Kinematic Positioning (RTK) sensor with centimetre-level accuracy and a widely used filter in both industry and academia for estimating states of nonlinear dynamic systems, the Extended Kalman Filter (EKF). MHE proves to be a better estimation algorithm than EKF when tracking trajectories over large distances, especially under highly noisy initial conditions. However, its performance declines in small-area environments and tight-curve maneuvers, where it becomes comparable to the EKF. Although the results did not fully meet the expectations set by simulations, the method outperforms traditional estimation approaches in non-ideal environments by leveraging a priori information in the form of optimization constraints.

Glosario

C++ Un lenguaje de programación compilado, de propósito general, que permite programación orientada a objetos y de bajo nivel.. 13

Estados internos Conjunto de variables que describen completamente el comportamiento de un sistema dinámico en un momento dado.. 2

GNSS El Sistema Global de Navegación por Satélite (GNSS, por sus siglas en inglés) es una tecnología que permite determinar la posición geográfica de un objeto en cualquier parte del mundo mediante señales emitidas por una constelación de satélites. Ejemplos de sistemas GNSS incluyen el GPS (Estados Unidos), GLONASS (Rusia), Galileo (Unión Europea) y BeiDou (China).. 11

IMU La Unidad de Medición Inercial (IMU, por sus siglas en inglés) es un dispositivo que combina sensores de acelerómetro, giroscopio y magnetómetro, para medir la velocidad angular, la aceleración lineal y la orientación de un objeto en el espacio.. 11

IoT El Internet de las Cosas (IoT, por sus siglas en inglés) se refiere a la interconexión de dispositivos físicos, vehículos, electrodomésticos y otros objetos integrados con sensores, software y conectividad de red, que les permite recolectar e intercambiar datos. El IoT permite la automatización y el control remoto de dispositivos, mejorando la eficiencia y facilitando la toma de decisiones en tiempo real.. 1

Python Un lenguaje de programación interpretado, de alto nivel y de propósito general, conocido por su sintaxis clara y legible.. 13

Slack En un algoritmo de optimización, una variable de slack es una variable adicional introducida para transformar restricciones de desigualdad en igualdades. Se utiliza para penalizar o minimizar el exceso de la restricción, facilitando la formulación y solución del problema de optimización.. 7

Índice de contenidos

Lista de Tablas

Lista de Figuras

1

Introducción

En la era actual, marcada por el acelerado avance tecnológico, la creciente sensorización, el Internet de las Cosas (IoT) y la integración de modelos de inteligencia artificial y aprendizaje maquinal, el procesamiento de datos se ha convertido en un tema crucial para las aplicaciones modernas. Este contexto motiva a trabajar hacia un mejor aprovechamiento de las mediciones disponibles, optimizando el uso de la información, factor esencial para la automatización y el desarrollo de la creciente Industria 4.0.

El presente trabajo se enfoca en la estimación de estados de vehículos terrestres utilizando mediciones redundantes de GPS de bajo y alto desempeño, combinando datos de sensores de distintas tecnologías para lograr mayor precisión a un costo reducido, lo cual resulta especialmente útil en aplicaciones de robótica móvil en entornos exteriores. Adicionalmente, se realiza una comparación directa entre la Estimación de Horizonte Móvil (MHE) y el Filtro Extendido de Kalman (EKF). El EKF es la adaptación a sistemas no-lineales del filtro de Kalman original postulado en 1960 por Rudolf E. Kalman [1], el cual ha sido ampliamente empleado en la industria aero-espacial, siendo crucial en el programa Apolo de la NASA [2], convirtiéndose su uso en un estándar para la navegación satelital.

A diferencia del filtro de Kalman, MHE permite agregar restricciones al cálculo de la estimación, así como tomar en cuenta la dinámica del modelo del sistema de un conjunto de muestras almacenadas con el tiempo al resolver un problema de optimización. Por otro lado la principal desventaja que presenta MHE es que el procesamiento necesario para solucionar el problema crece linealmente con la cantidad de muestras almacenadas como se explica en [3], pero dado el avance en el poder de cómputo de forma sostenida a través de los años [4] es que este apartado ya no es un problema.

2

Estado del arte

2.1 Sistema

Para la estimacion de estados del sistema es necesario definir el sistema a modelar y estimar, teniendo claras las entradas del sistema, las variables físicas a observar, y su dinámica en base al libro [5].
El vehículo es un Husky A200 de Clearpath Robotics, el cual es un robot diferencial de 4 ruedas todo terreno, diseñado para aplicaciones de investigación y desarrollo en robótica móvil.

En este trabajo se busca estimar la posición del vehículo en un plano tangencial al suelo, por lo que las variables de estado a estimar son la posición (x, y) y la orientación θ del vehículo respecto al eje x de dicho plano de referencia global.

$$Q = \begin{bmatrix} Q_{ang} \\ Q_x \\ Q_y \end{bmatrix} = \chi = \begin{bmatrix} \theta \\ x \\ y \end{bmatrix} \tag{2.1}$$

2.1.1 Modelamiento del vehículo terrestre autónomo

El modelo cinemático del robot consiste en como el vehículo actúa sobre si mismo para modificar el estado Q. Dado que el robot es un vehiculo diferencial de 4 ruedas simétricas, la actuación de estas puede simplificarse al de 2 ruedas virtuales ubicadas en el centro del vehículo, una a la izquierda y otra a la derecha, las cuales son controladas de forma independiente.

Además, el vehículo presenta controladores internos de torque para seguimiento de velocidad angular, lo cual permite modelar la dinámica del vehículo sólo con actuación de velocidades angulares de la ruedas izquierda y derecha, denotadas como $\dot{\theta}_{r_{izq}}$ y $\dot{\theta}_{r_{der}}$ respectivamente.

Luego se debe considerar los diferentes planos de referencia, primero, respecto al robot diferencial, este sólo puede desplazarse en la direcciónn frontal permitida por sus ruedas y chásis rígido, es decir posee la siguiente dinámica interna con R como el radio de las ruedas.

$$\dot{Q}_{robot} = \begin{bmatrix} 0 \\ \frac{R}{2}(\dot{\theta}_{r_{der}} + \dot{\theta}_{r_{izq}}) \\ 0 \end{bmatrix} \tag{2.2}$$

Al cambiar al plano de referencia global, es necesario considerar la orientación del robot Q_{ang} para proyectar el desplazamiento frontal en las direcciones x e y, donde ademas se considera la rotación del robot dada por la diferencia de velocidades angulares de las ruedas dividida por la distancia entre estas L.

$$\begin{bmatrix} \dot{Q}_{ang} \\ \dot{Q}_x \\ \dot{Q}_y \end{bmatrix} = \begin{bmatrix} \frac{R}{L}(\dot{\theta}_{r_{der}} - \dot{\theta}_{r_{izq}}) \\ \frac{R}{2}(\dot{\theta}_{r_{der}} + \dot{\theta}_{r_{izq}})Cos(Q_{ang}) \\ \frac{R}{2}(\dot{\theta}_{r_{der}} + \dot{\theta}_{r_{izq}})Sen(Q_{ang}) \end{bmatrix} = \frac{R}{2} \begin{bmatrix} \frac{2}{L} & -\frac{2}{L} \\ Cos(Q_{ang}) & Cos(Q_{ang}) \\ Sen(Q_{ang}) & Sen(Q_{ang}) \end{bmatrix} \begin{bmatrix} \dot{\theta}_{r_{der}} \\ \dot{\theta}_{r_{izq}} \end{bmatrix} \tag{2.3}$$

Entonces la ecuación de estado del vehículo queda definida por 2.4 con $\omega(t)$ como el ruido del proceso inherente al sistema.

$$Q(t) = Q_0 + t\frac{R}{2} \begin{bmatrix} \frac{2}{L} & -\frac{2}{L} \\ Cos(Q_{ang}(t)) & Cos(Q_{ang}(t)) \\ Sen(Q_{ang}(t)) & Sen(Q_{ang}(t)) \end{bmatrix} \begin{bmatrix} \dot{\theta}_{r_{der}}(t) \\ \dot{\theta}_{r_{izq}}(t) \end{bmatrix} + \omega(t) \tag{2.4}$$

En donde se nota la no-linealidad del sistema debido a que $Q(t)$ depende de las funciones trigonométricas de su propio estado de orientación $Q_{ang}(t)$.

2.2 Estimación de estados

La estimacion de estados corresponde al cálculo de variables de estado del sistema por medio de mediciones realizadas con sensores o modelos matemáticos del sistema, considerando que las mediciones pueden ser afectadas por perturbaciones externas o internas del sistema, y que el modelo matemático puede no represantar de forma exacta la dinámica del sistema.

2.2.1 Kalman

El filtro de Kalman aplicado a estimación de estados es un algoritmo que toma mediciones directas o indirectas sobre un sistema dinámico y permite calcular una aproximación de sus Estados internos mas precisa que las obtenidas inicialmente por las mediciones al considerar un modelo lineal del sistema y ruidos gaussianos no correlacionados.

Este filtro se encuentra entre los algoritmos de fusión de datos más importantes y comunes hoy en día, se puede encontrar en dispositivos de uso diario como celulares, automóviles, dispositivos de navegación satelital, etc. dado que es un estimador óptimo para sistemas lineales afectados por ruidos gaussianos, pero sigue siendo muy robusto cuando no se dan las condiciones perfectas y su baja complejidad algorítmica permite aplicaciones en tiempo real y de formas muy diversas, La variante mas apropiada para el sistema a estimar es el filtro de Kalman extendido. [6]

El filtro de Kalman extendido (EKF) es una extensión del Filtro de Kalman estándar que permite su aplicación a sistemas no lineales. Aproxima el sistema no lineal mediante una linealización alrededor del estado actual, utilizando una expansión de series de Taylor de primer orden, y consta de dos etapas.
La etapa de predicción consiste en estimar (las variables estimadas se denotan con el símbolo "ˆ") el estado del sistema actual $\hat{x}_k^-$ que debería tener teóricamente según un modelo de la forma $f(x_k, u_k)$ en donde x_k es el vector de estados del sistema y u_k el vector de perturbaciones en un instante k, por lo que es necesario tener conocimiento de un estado anterior $k-1$ y las perturbaciones ocurridas en ese instante. Además también se estima la matriz P_k^- que corresponde a la covarianza del error inducido al predecir el estado actual del sistema, es decir, cuantifica que tan confiable es el modelo, y por tanto que tan certera es la predicción $\hat{x}_k^-$ realizada.

La segunda etapa de corrección se encarga de compensar la estimación realizada con el modelo, con una medición ponderada del estado del sistema.La ponderación K_k depende de que tan grande sea la covarianza introducida al evolucionar y medir el sistema. Posteriormente se actualiza la covarianza P_k al agregar a P_k^- la información obtenida al medir la muestra.

Etapa de Predicción

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1})$$

$$P_k^- = F_{k-1} P_{k-1} F_{k-1}^T + Q_{k-1}$$

Donde:

- $\hat{x}_k^-$: Estado predicho en el instante k por el modelo.
- f: Función no lineal que describe la dinámica del sistema.
- $\hat{x}_{k-1}$: Estimación del estado en el instante $k-1$.
- u_{k-1}: Vector de entradas de control en el instante $k-1$.
- P_k^-: Covarianza del error predicha.
- F_{k-1}: Matriz jacobiana de f respecto a x, evaluada en $\hat{x}_{k-1}$.
- P_{k-1}: Covarianza del error en el instante $k-1$.
- Q_{k-1}: Covarianza del ruido del proceso.

Etapa de Corrección

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-))$$

$$P_k = (I - K_k H_k) P_k^-$$

Donde:

- K_k: Ganancia de Kalman en el instante k.
- H_k: Matriz jacobiana de la función de observación h, evaluada en $\hat{x}_k^-$.
- R_k: Covarianza del ruido de las mediciones.

- $\hat{x}_k$: Estimación actualizada del estado en el instante k.
- z_k: Medición real en el instante k.
- h: Función no lineal que relaciona los estados con las mediciones.
- P_k: Covarianza del error actualizada.
- I: Matriz identidad.

2.2.2 MHE

Tal y como se detalla en [7] MHE consiste en optimizar una función objetivo dada por la ecuación 2.5:

$$V_N(\chi, \omega, v, \bar{x}) := \rho V_p(\chi, \bar{x}) + \sum_{k=0}^{N-1} \ell(\omega(k), v(k)) \tag{2.5}$$

El horizonte móvil corresponde a la cantidad de estimaciones de estados a realizar por el MHE, cuyo tamaño esta dado por N.

$\ell(\omega(k), v(k))$ puede tomar distintas definiciones, en este trabajo se utiliza su forma estándar definida por la siguiente expresión con los parámetros a ajustar M_Q y M_R, ambas matrices cuadradas del tamaño de la cantidad de estados a estimar y definidas positivas, que ponderan los ruidos de proceso y medición del sistema. Se nota además que en un sistema lineal con ruidos gaussianos no-correlacionados estas matrices corresponderían a la diagonal de la inversa de sus covarianzas:

$$\ell(\omega(k), v(k)) = \omega(k)^T M_R\, \omega(k) + v(k)^T M_Q\, v(k) \tag{2.6}$$

La expresión $\rho V_p(\chi, \bar{x})$ es conocida como costo de arribo, consiste en la ponderación dada a la estimación fuera del horizonte móvil, de esta manera no se elimina información importante para la estimación. En la iniciación del algoritmo es interpretada como las condiciones iniciales del sistema, mientras que en las siguientes iteraciones corresponde a la información descartada por el horizonte móvil.En este trabajo la expresión de costo de arribo esta dado por:

$$\rho V_p(\chi, \bar{x}) = (x_{k-N} - \hat{x}_{k-N})^T P\, (x_{k-N} - \hat{x}_{k-N}) \tag{2.7}$$

En donde P es una matriz cuadrada del tamaño de la cantidad de estados a estimar y definida positiva, se calcula como la covarianza del error actualizado P_k anteriormente expresada en el filtro de Kalman

extendido, llamado costo de arribo de suavizado como muestra la figura 2.1.

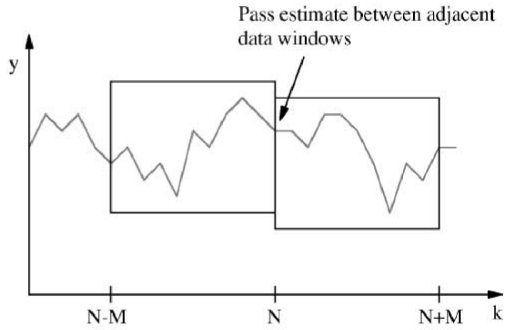


Figura 2.1: Visualización de filtrado[8].

Finalmente, el problema se puede expresar como la minimización de los error de estimación, al encontrar los ruidos que mejor se ajusten al problema planteado:

$$\min_{\chi,\omega,v} V_N(\chi(t-N),\omega,v,\bar{x})$$

Donde es necesario definir como mínimo una restricción que establezca el modelo de la dinámica del sistema y como es afectada por el ruido del proceso ω con $f(\chi,\omega)$, y también se debe definir como las mediciones se relacionan con los estados internos del sistema χ y el ruido de medición v con $y = h(\chi)+v$.

$$s.t.\,\chi^{+} = f(\chi,\omega) \tag{2.8}$$

$$y = h(\chi)+v \tag{2.9}$$

En donde se pueden agregar mas restricciones si se tiene mayor conocimiento del sistema, que en el caso de este trabajo se establecen como el conocimiento de la geometría del robot y la disposición de los GPS. En este caso la ecuación (2.9) se reemplaza por la restricción (2.10), en donde se tiene un vector de dimensión 9, pero como el vector de estado Q a estimar del robot es de 3 dimensiones, es necesario reducirlo, esto se logra manteniendo el primer estado correspondiente a la dirección del robot con (2.11), mientras que el resto de dimensiones se reducen a través de las restricciones de redundancia especificadas en (2.12). Además se cuenta con las restricciones geométricas de la ecuación (2.13) que permiten reducir la incertidumbre entre las mediciones.

$$\begin{bmatrix} \hat{q}_{\text{ang}} \\ \hat{q}_{x_0} \\ \hat{q}_{y_0} \\ \hat{q}_{x_1} \\ \hat{q}_{y_1} \\ \hat{q}_{x_2} \\ \hat{q}_{y_2} \\ \hat{q}_{x_3} \\ \hat{q}_{y_3} \end{bmatrix} = \begin{bmatrix} h_{\text{ang}} \\ h_{x0} \\ h_{y0} \\ h_{x1} \\ h_{y1} \\ h_{x2} \\ h_{y2} \\ h_{x3} \\ h_{y3} \end{bmatrix} + \begin{bmatrix} v_{\text{ang}} \\ v_{x_0} \\ v_{y_0} \\ v_{x_1} \\ v_{y_1} \\ v_{x_2} \\ v_{y_2} \\ v_{x_3} \\ v_{y_3} \end{bmatrix} \tag{2.10}$$

$$\hat{Q}_{ang} = \hat{q}_{\text{ang}} = h_{\text{ang}} + v_{\text{ang}} \tag{2.11}$$

$$\begin{bmatrix} \hat{Q}_x \\ \hat{Q}_y \end{bmatrix} = \begin{bmatrix} \hat{q}_{x_0} \\ \hat{q}_{y_0} \end{bmatrix} + \text{GPS}_{dist_0} \begin{bmatrix} Cos(h_{\text{ang}} + \text{GPS}_{\text{ang}_0}) \\ Sen(h_{\text{ang}} + \text{GPS}_{\text{ang}_0}) \end{bmatrix} + \begin{bmatrix} S_{x_0} \\ S_{y_0} \end{bmatrix} \tag{2.12}$$

$$\begin{bmatrix} \hat{Q}_x \\ \hat{Q}_y \end{bmatrix} = \begin{bmatrix} \hat{q}_{x_1} \\ \hat{q}_{y_1} \end{bmatrix} + \text{GPS}_{dist_1} \begin{bmatrix} Cos(h_{\text{ang}} - \text{GPS}_{\text{ang}_1}) \\ Sen(h_{\text{ang}} - \text{GPS}_{\text{ang}_1}) \end{bmatrix} + \begin{bmatrix} S_{x_1} \\ S_{y_1} \end{bmatrix}$$

$$\begin{bmatrix} \hat{Q}_x \\ \hat{Q}_y \end{bmatrix} = \begin{bmatrix} \hat{q}_{x_2} \\ \hat{q}_{y_2} \end{bmatrix} + \text{GPS}_{dist_2} \begin{bmatrix} Sen(h_{\text{ang}} + \text{GPS}_{\text{ang}_2}) \\ -Cos(h_{\text{ang}} + \text{GPS}_{\text{ang}_2}) \end{bmatrix} + \begin{bmatrix} S_{x_2} \\ S_{y_2} \end{bmatrix}$$

$$\begin{bmatrix} \hat{Q}_x \\ \hat{Q}_y \end{bmatrix} = \begin{bmatrix} \hat{q}_{x_3} \\ \hat{q}_{y_3} \end{bmatrix} + \text{GPS}_{dist_3} \begin{bmatrix} Sen(h_{\text{ang}} - \text{GPS}_{\text{ang}_3}) \\ -Cos(h_{\text{ang}} - \text{GPS}_{_3}) \end{bmatrix} + \begin{bmatrix} S_{x_3} \\ S_{y_3} \end{bmatrix}$$

$$\begin{bmatrix} (\hat{q}_{x_0} - \hat{q}_{x_2})^2 + (\hat{q}_{y_0} - \hat{q}_{y_2})^2 \\ (\hat{q}_{x_1} - \hat{q}_{x_3})^2 + (\hat{q}_{y_1} - \hat{q}_{y_3})^2 \end{bmatrix} = \begin{bmatrix} \text{GPS}_{\text{norm}_{0\text{-}2}}{}^2 \\ \text{GPS}_{\text{norm}_{1\text{-}3}}{}^2 \end{bmatrix} + \begin{bmatrix} S_{\text{norm}_{0\text{-}2}} \\ S_{\text{norm}_{1\text{-}3}} \end{bmatrix} \tag{2.13}$$

Donde para $i \in \{0,1,2,3\}$:

- $\hat{Q}$: Vector de estado estimado, definido como

$$\hat{Q} = \begin{bmatrix} \hat{Q}_{\text{ang}} \\ \hat{Q}_x \\ \hat{Q}_y \end{bmatrix}$$

- $\hat{Q}_{\text{ang}}$: Estimación de la dirección del robot.
- $\hat{Q}_x$, $\hat{Q}_y$: Estimación de la posición del robot.
- $\hat{q}_{x_i}$, $\hat{q}_{y_i}$: Estimación de la posición del sensor GPS i.
- h_{ang}: Medición de la dirección del robot.
- h_{x_i}, h_{y_i}: Medición de la posición del GPS i.
- v_{ang}: Ruido en la medición del ángulo.
- v_{x_i}, v_{y_i}: Ruido en las mediciones de posición del GPS i.
- S_{x_i}, S_{y_i}: Variables de Slack para la redundancia de los GPS i.
- GPS_{dist_i}: Parámetro que define la distancia entre el GPS i y el centro del robot.
- $\text{GPS}_{\text{ang}_i}$: Parámetro que define el ángulo de posicionamiento del GPS i respecto al chasis del robot.
- $\text{GPS}_{\text{norm}_{0\text{-}2}}$: Parámetro que define la distancia entre los GPS 0 y 2.
- $\text{GPS}_{\text{norm}_{1\text{-}3}}$: Parámetro que define la distancia entre los GPS 1 y 3.
- $S_{\text{norm}_{0\text{-}2}}$: Variable de Slack para la restricción geométrica entre los GPS 0 y 2.
- $S_{\text{norm}_{1\text{-}3}}$: Variable de Slack para la restricción geométrica entre los GPS 1 y 3.

Dadas las investigaciones en el modelo de estimación que motiva este proyecto, se ha estudiado lo expuesto por J.B. Rawlings, D.Q. Mayne, and M. Diehl [7], y como muestra la figura 2.2, el estado del arte indica que MHE es la herramienta correcta para mejorar las mediciones en entornos no lineales respecto al EKF, dado permite incorporar restricciones al algoritmo de optimización que permita aún mayor precisión.

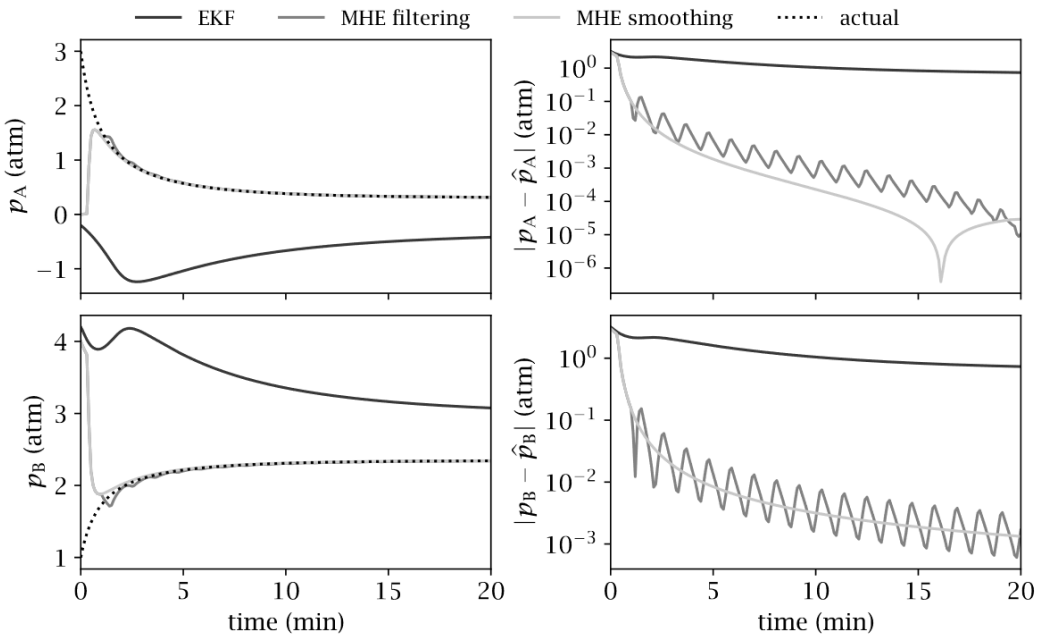


Figura 2.2: Visualización de comparación de costos de arribo y EKF[7].

3

Desarrollo de la solución

3.1 Hardware

3.1.1 Husky A200

El Husky A200 [9] presenta soporte modular para sensores e integración con ROS dada por el fabricante. EL robot es ilustrado en la figura 3.1 sin sensores ni barras de soporte. Se cuenta con las especificaciones en la tabla 3.1.

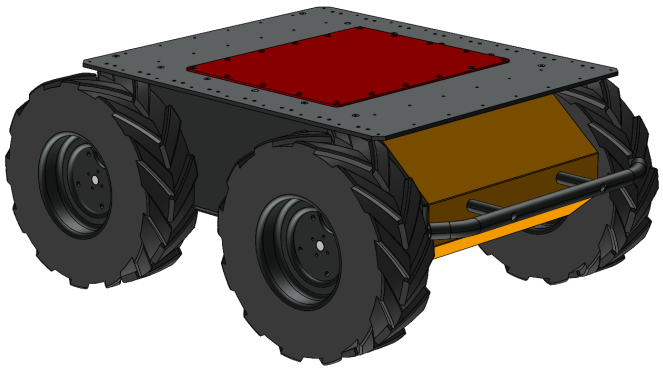


Figura 3.1: Representación Husky A200 [9].

Los sensores cuentan con la disposición ilustrada en la figura 3.2, en donde 0 y 1 en amarillo son GPS de baja calidad, 2 y 3 en rojo son GPS de buena calidad, 2 y 3 en azul son las IMU.

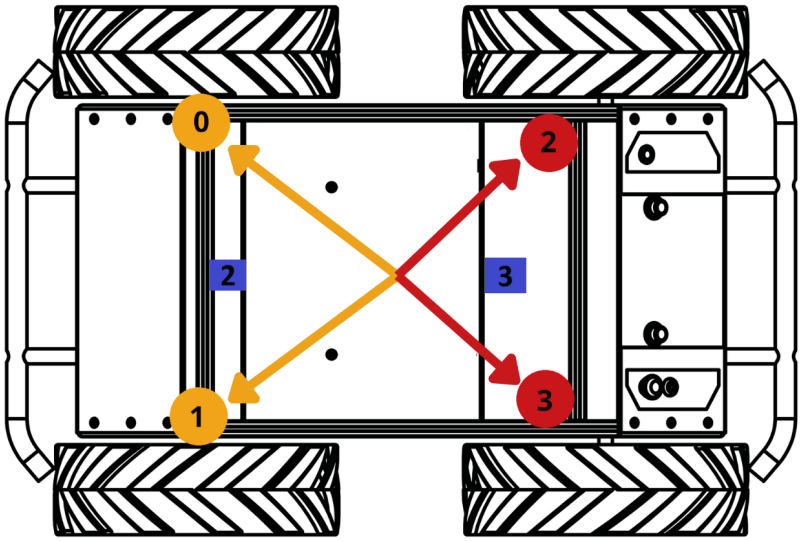


Figura 3.2: Disposición de sensores en robot Husky A200.

Especificación	**Medida**
Dimensión, Longitud	990 mm
Dimensión, Ancho	670 mm
Dimensión, Altura	390 mm
Ancho de vía	550 mm
Distancia entre ejes	512 mm
Distancia al suelo	130 mm
Masa	50 kg
Velocidad máxima	1 m/s
Tiempo de operación, Promedio	3 horas
Encoders de rueda	78,000 ticks/m

Tabla 3.1: Especificaciones Husky A200

3.1.2 Sensores GPS

El primer par de sensores 0 y 1 corresponden al modelo U-box NEO-7M[10] montado en un modulo HG-C02, es un GNSS diseñado para uso en drones de bajo presupuesto e incorpora su propia antena como muestra la figura 3.3, y cuenta con las especificaciones de la tabla 3.2. También cuenta con sensores IMU, que no se usaran en este proyecto.

Figura 3.3: Módulo U-box Neo-M7.

Característica	**Especificación**
Precisión	2.5 m (CEP)
Frecuencia de actualización	1 - 10 Hz
Sensibilidad (Rastreo/Adquisición)	-167 dBm / -148 dBm
Número de canales	56 canales GNSS
Comunicación	UART
Formato	NMEA

Tabla 3.2: Especificaciones del módulo u-blox NEO-7M

El segundo par de sensores a usar corresponde a VectorNav VN200 [11], es un GNSS miniatura y de alto desempeño, diseñado para aplicaciones demandantes. Su empaquetado se ilustra en la figura 3.4 y sus especificaciones en la tabla 3.3

3.1.3 Sensor IMU

Este sensor es incluido en el mismo paquete que el VN200 de la figura 3.4, por lo tanto se cuenta con dos IMU disponibles para realizar las mediciones de orientación. Las especificaciones relevantes se encuentran

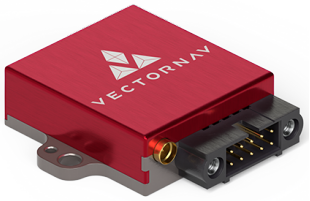


Figura 3.4: Módulo VN200 [11].

Característica	Especificación
Precisión	1.0 m (horizontal) / 1.5 m (vertical) RMS
Frecuencia de actualización	5 Hz
Sensibilidad (Rastreo/Adquisición)	-159 dBm / -138 dBm
Número de canales	72 canales GNSS
Comunicación	Serial - USB
Formato	Propietario VectorNav

Tabla 3.3: Especificaciones del módulo VN-200 GNSS

detalladas en la tabla 3.4.

Característica	Especificación
Rango de medida	±2000 °/s
Estabilidad de bias en operación	5-10 °/hr
Densidad de ruido	0.0035 °/s/$\sqrt{\text{Hz}}$
Frecuencia de actualización	800 Hz
Sensibilidad cruzada	0.05°
Resolución	0.02 °/s

Tabla 3.4: Especificaciones del giroscopio del módulo VN-200

3.1.4 Sensor RTK

Para la validación de posición es usado el sensor GNSS RTK Navcom SF-3040 [12]. Este cuenta con dos módulos como el ilustrado en la figura 3.5, uno se debe posicionar en un punto fijo, actuando como estación base, y el otro se monta en el robot, actuando como estación móvil, de esta manera al contar con una doble medición logra corregir gran parte del error, estableciendo una medición fidedigna con error de centímetros.

Figura 3.5: Sensor Navcom SF-3040 [12].

Característica	Especificación
Precisión	1 cm + 0.5 ppm
Frecuencia de actualización	5 - 10 Hz
Sensibilidad (Rastreo/Adquisición)	-112dBm
Número de canales	66 canales GNSS
Comunicación	serial - USB
Formato	NMEA

Tabla 3.5: Especificaciones del receptor GNSS NavCom SF-3040

3.2 Software

3.2.1 ROS

Robot Operating System (ROS) [13] es un framework de código abierto utilizado para el desarrollo de software en robótica. Proporciona herramientas, bibliotecas y convenciones para simplificar la creación de aplicaciones al permitir una alta reutilización de código con su sistema de paquetes. Actúa como interfaz entre las mediciones realizadas por sensores y mensajes de actuación calculados por un controlador, esto se logra con la creación de nodos que publican información en un tópico ROS y nodos que escuchan la información publicada en el tópico, de esta manera el código y los sensores se comunican a través de nodos, cuyo comportamiento esta determinado por paquetes con código en lenguaje Python o C++, en la tabla 3.6 se especifican los paquetes que se utilizan y una breve descripción.

Paquete	Descripción
husky_base	Activa sub-paquetes para comunicarse con el robot, creando tópicos de muestreo de sensores y actuaciones en el Husky.
navsat	Permite la lectura de sensores de posición en formato serial NMEA conectados por USB. Es usado para muestrear los GNSS Neo-M7 y SF-3040
vectornav	Permite la lectura de sensores de posición en formato serial VectorNav conectados por USB. Es usado para muestrear los GNSS VN200
parches	Paquete desarrollado por investigadores del AC3E, permite movimientos suaves al hacer upsampling de actuaciones de baja frecuencia

Tabla 3.6: Especificaciones de paquetes ROS.

3.2.2 Matlab

Es un software con un entorno gráfico con su propio lenguaje de programación enfocado en el cómputo numérico desarrollado por MathWorks [14]. Dada su versatilidad, su gran cantidad de Toolbox y su

estructura de alto nivel es que Matlab se ha popularizado en áreas de ingeniería, ciencia y matemática. Al contar con integración de ROS es posible utilizar código que lea tópicos directamente en tiempo real, y posteriormente enviar comandos de movimiento al robot Husky. Se ha elegido como la herramienta de implementación lógica y de cómputo del algoritmo de estimación MHE dada su fácil implementación que a su vez permite compilación del código en lenguaje C, de esta manera se logra una ejecución eficiente en tiempo real.

3.2.3 CasADi

Es una herramienta de código abierto especializada en implementación de métodos numéricos para la optimización no-lineal y la resolución de ecuaciones diferenciales [15]. Permite trabajar con gran versatilidad al hacer uso de variables simbólicas y resolución de problemas de optimización en tiempo real. Una vez definido el problema de optimización con CasADi, se utiliza el solver de punto interior“IPOPT" para calcular las estimaciones del MHE y actuaciones del MPC respectivamente, dado que, en comparación con otras herramientas de optimización, este se encuentra diseñado para problemas no-lineales y además cuenta con un robusto comportamiento ante la adición de restricciones como se discute en [16].

4

Implementación

4.1 Pruebas virtuales

En este capítulo se desarrolla la implementación de los algoritmos de MHE y MPC de manera virtual en Matlab. El objetivo de realizar una simulación de la dinámica del robot, realizando pruebas virtuales que validen las técnicas discutidas anteriormente de manera teórica en un entorno lo más cercano a la realidad posible, de esta manera se ajustan los parámetros y se corrigen los errores que puedan surgir de manera mas eficiente al concentrar el trabajo sólo en un buen desarrollo de los algoritmos. El flujo de información de simulación se muestra en la figura 4.1.

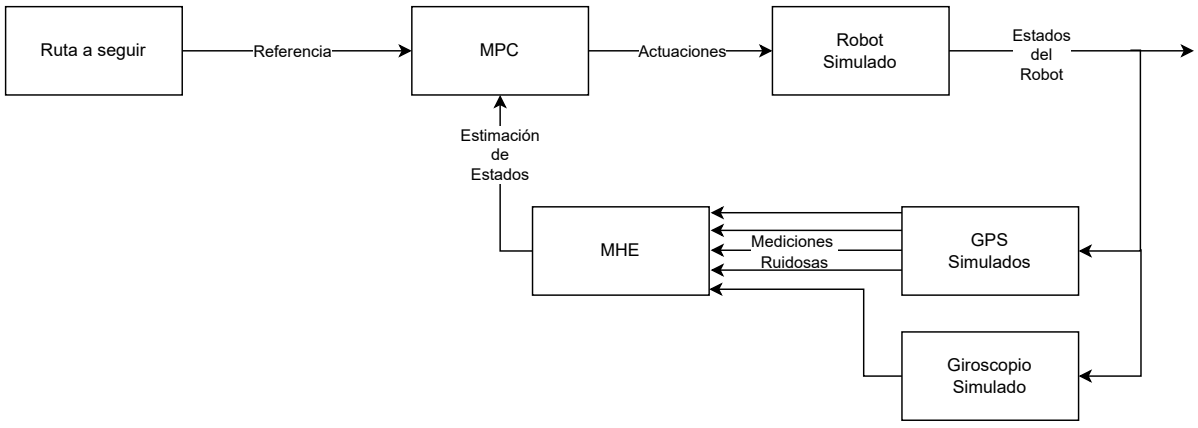


Figura 4.1: Esquema de control de implementación virtual

4.1.1 MHE

El algoritmo de MHE es implementado en el código Matlab 4.1:

```
Nh = 20;
optiMHE = casadi.Opti();

qMHE = optiMHE.variable(nq, Nh + 1);
wMHE = optiMHE.variable(nq, Nh);
vMHE = optiMHE.variable(nq, Nh + 1);

alphaMHE = optiMHE.variable(4, Nh+1);

y_meas = optiMHE.parameter(4 * 2 + 1, Nh + 1);
y_meas_convx = optiMHE.variable(3, Nh + 1);

uPast = optiMHE.parameter(nu, Nh);
X = optiMHE.variable(nq);
xBar = optiMHE.parameter(nq);
Pmhe = optiMHE.parameter(nq,nq);
JmhePast = optiMHE.parameter(1);

Rmhe = diag([1/gps_noise_factor_variance, 1/gps_noise_factor_variance, 1/gyro_noise_factor_variance])*0.15;
Qmhe = diag([1, 1, 1/(2*pi)] * 800);

Jmhe = X.' * Pmhe * X + JmhePast;
optiMHE.subject_to(X == qMHE(:, 1) - xBar);

for k = 1:Nh
    Jmhe = Jmhe + vMHE(:, k).' * Rmhe * vMHE(:, k) + wMHE(:, k).' * Qmhe * wMHE(:, k);
    optiMHE.subject_to(qMHE(:, k + 1) == F(qMHE(:, k), uPast(:, k), Ts, w, wMHE(:, k)));

    R = [cos(y_meas(4 * 2 + 1, k)), -sin(y_meas(4 * 2 + 1, k));
         sin(y_meas(4 * 2 + 1, k)),  cos(y_meas(4 * 2 + 1, k))];

    optiMHE.subject_to(sum(alphaMHE(:,k))==1);
    optiMHE.subject_to((0<=alphaMHE(:,k))<=1);

    y_meas_convx (:,k)= [(y_meas(1:2, k) - R * 0.5 * [lg; -wg]) * alphaMHE(1, k) + ...
                         (y_meas(3:4, k) - R * 0.5 * [lg; wg]) * alphaMHE(2, k) + ...
                         (y_meas(5:6, k) - R * 0.5 * [-lg; wg]) * alphaMHE(3, k) + ...
                         (y_meas(7:8, k) - R * 0.5 * [-lg; -wg]) * alphaMHE(4, k); y_meas(4 * 2 + 1, k)];

    optiMHE.subject_to(qMHE(:, k) + vMHE(:, k) == (y_meas_convx (:,k)));

end

R = [cos(y_meas(4 * 2 + 1, Nh + 1)), -sin(y_meas(4 * 2 + 1, Nh + 1));
     sin(y_meas(4 * 2 + 1, Nh + 1)),  cos(y_meas(4 * 2 + 1, Nh + 1))];

optiMHE.subject_to(sum(alphaMHE(:,Nh+1))==1);
optiMHE.subject_to((0<=alphaMHE(:,Nh+1))<=1);

y_meas_convx(:,k) = [(y_meas(1:2, Nh + 1) - R * 0.5 * [lg; -wg]) * alphaMHE(1, Nh + 1) + ...
                (y_meas(3:4, Nh + 1) - R * 0.5 * [lg; wg]) * alphaMHE(2, Nh + 1) + ...
                (y_meas(5:6, Nh + 1) - R * 0.5 * [-lg; wg]) * alphaMHE(3, Nh + 1) + ...
                (y_meas(7:8, Nh + 1) - R * 0.5 * [-lg; -wg]) * alphaMHE(4, Nh + 1); y_meas(4 * 2 + 1, Nh + 1)];

optiMHE.subject_to(qMHE(:, Nh + 1) + vMHE(:, Nh + 1) == y_meas_convx(:,Nh+1) );

Jmhe = Jmhe + vMHE(:, Nh + 1).' * Rmhe * vMHE(:, Nh + 1);

optiMHE.minimize(Jmhe);

optiMHE.subject_to((-gps_noise_factor*3 <= wMHE) <= gps_noise_factor*3 );
optiMHE.subject_to((-gps_noise_factor*3 <= vMHE) <= gps_noise_factor*3 );

p_optsMHE = struct('expand', true);
s_optsMHE = struct('sb', 'yes', 'print_level', 0, 'gamma_theta', 1e-2, 'jacobian_approximation', 'exact', ...
'fast_step_computation', 'yes', 'warm_start_init_point', 'yes');
optiMHE.solver('ipopt', p_optsMHE, s_optsMHE);
```

Código 4.1: Implementación MHE en simulación

4.1.2 MPC

El algoritmo de MPC es implementado en el código Matlab 4.2:

```
Nc = 30;

max_angular_velocity = vel_max / wheel_radius;
wheel_vel = max_angular_velocity / 2;

U_lb = [-wheel_vel; -wheel_vel];
U_ub = [wheel_vel; wheel_vel];

dU_lb = U_lb * 0.1;
dU_ub = U_ub * 0.1;

optiMPC = casadi.Opti();

qMPC = optiMPC.variable(nq, Nc + 1);
u = optiMPC.variable(nu, Nc);
qref = optiMPC.parameter(nr);
qinit = optiMPC.parameter(nq);
u_last = optiMPC.parameter(nu);

Wmpc = diag([1, 1, 2] * 2);
Rmpc = diag([0.08, 0.08]);

Jmpc = 0;
for k = 1:Nc + 1
    error = qMPC(:, k) - qref;

    Jmpc = Jmpc + (error).' * Wmpc * (error);
    if k < Nc + 1
        optiMPC.subject_to(qMPC(:, k + 1) == F(qMPC(:, k), u(:, k), Ts, w, zeros(3, 1)));

        if k ~= 1
            optiMPC.subject_to((dU_lb <= (u(:, k) - u(:, k - 1)) / Ts) <= dU_ub);
        else
            optiMPC.subject_to((dU_lb <= (u(:, k) - u_last) / Ts) <= dU_ub);
        end
        optiMPC.subject_to((U_lb <= u(:, k)) <= U_ub);
        Jmpc = Jmpc + u(:, k).' * Rmpc * u(:, k);
    end
end

optiMPC.subject_to(u(1, :) * 0.5 + u(2, :) * 0.5 >= 0);

optiMPC.minimize(sum(sum(u.^2)));

optiMPC.minimize(Jmpc);

optiMPC.subject_to(qMPC(:, 1) == qinit);

p_optsMPC = struct('expand', true);
s_optsMPC = struct('sb', 'yes', 'print_level', 0, 'warm_start_init_point', 'yes');
optiMPC.solver('ipopt', p_optsMPC, s_optsMPC);
```

Código 4.2: Implementación MPC en simulación

4.1.3 Resultados

Los resultados obtenidos de la simulación se pueden observar en la figura 4.2, esta corresponde al seguimiento por parte del robot (triángulo azul) de una lemniscata de 12 metros de ancho, a una velocidad de $1[m/s]$, correspondiente a la linea negra. Se puede observar como la linea de rayas rojas y la linea punteada magenta siguen logran estimar satisfactoriamente al acercarse a la linea azul. Sin embargo

las figuras 4.3 y 4.4 muestran como consistentemente MHE mejora la estimación realizada por el EKF, especialmente en el error de posición, esto se atribuye a que MHE maneja mejor los errores de condiciones iniciales muy distanciados de la realidad. Por su parte el algoritmo de MPC fue implementado exitosamente logrando actuar sobre las ruedas del robot para seguir la lemniscata de referencia como muestra la figura 4.5, en donde se nota que filtra el ruido de estimación del MHE y en las figuras 4.6 y 4.7 se puede observar el desempeño obtenido por ambos algoritmos en conjunto donde el control tiene un error de posición máximo de $0.5[m]$.

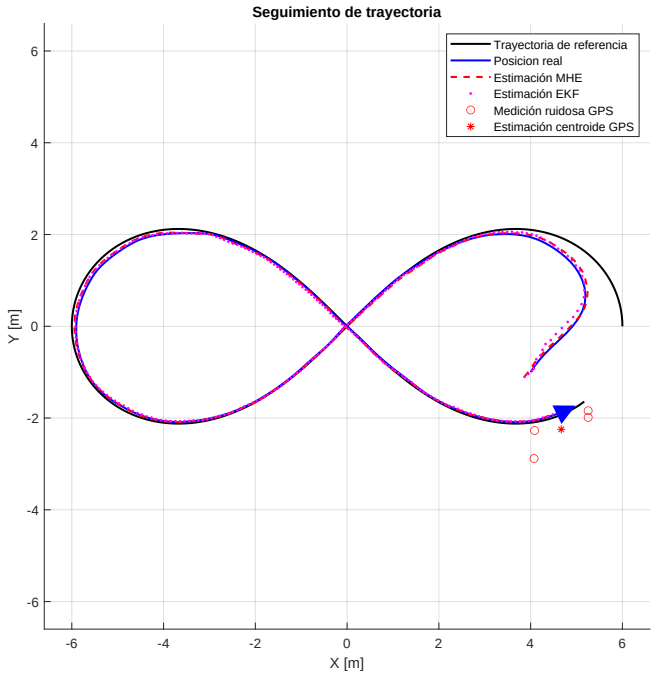


Figura 4.2: Gráfico de seguimiento de trayectoria virtual

Figura 4.3: Gráfico del error de estimación de posición

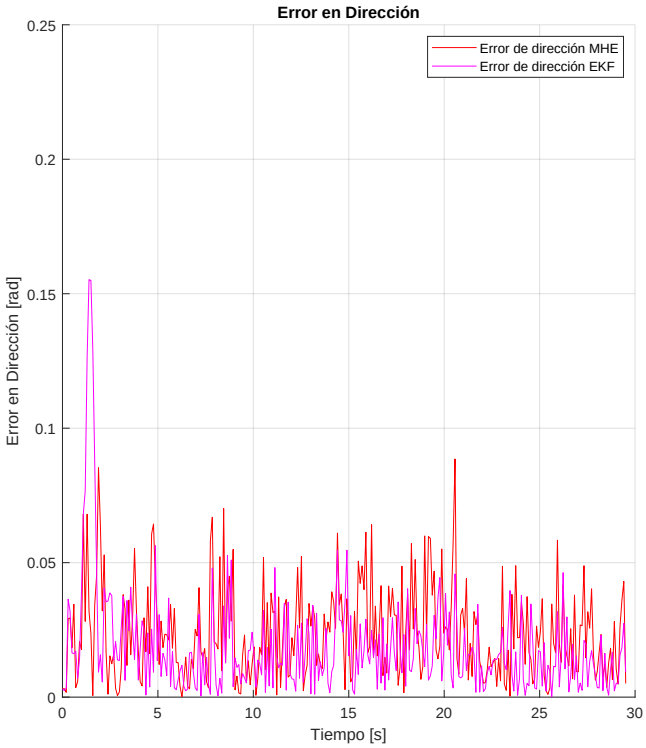


Figura 4.4: Gráfico del error de estimación de dirección

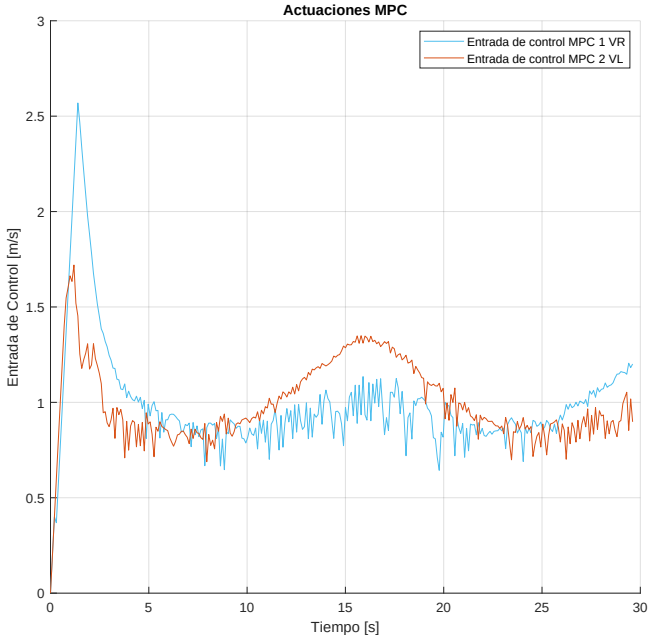


Figura 4.5: Gráfico de actuaciones sobre robot

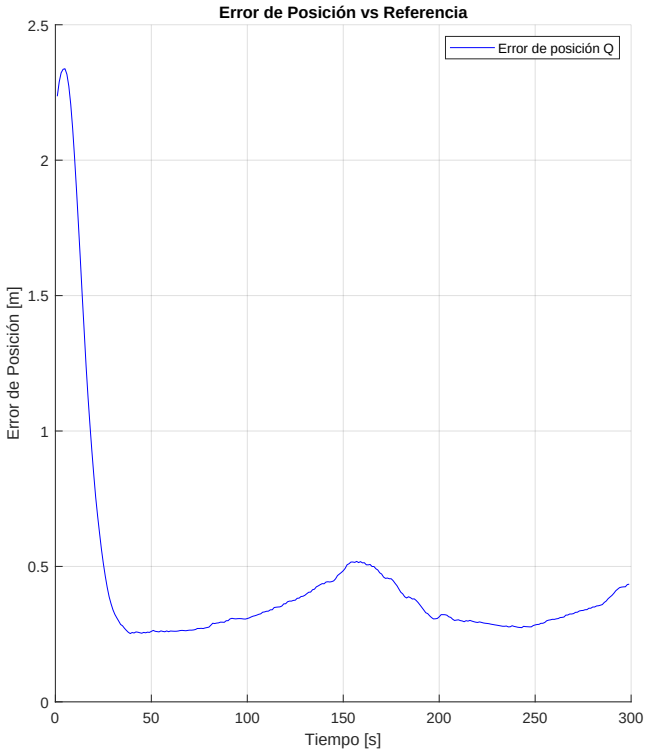


Figura 4.6: Gráfico del error de seguimiento de posición

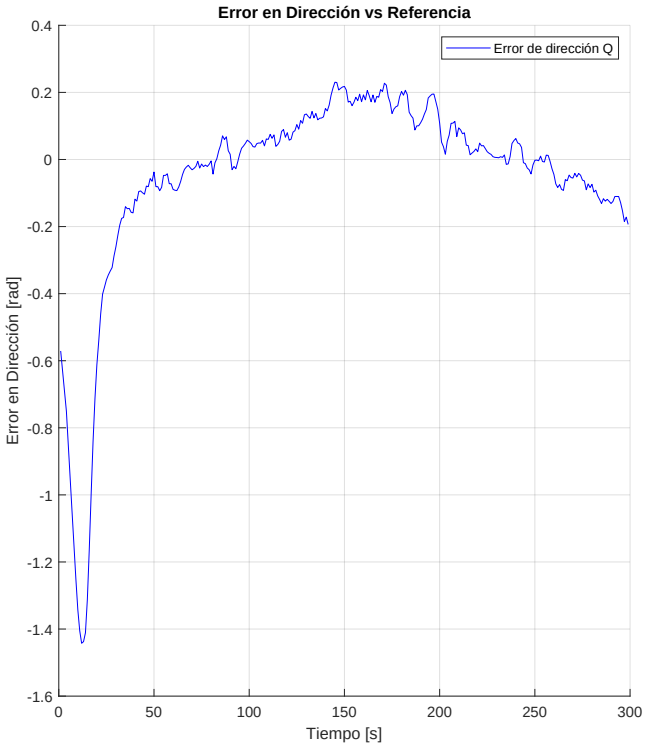


Figura 4.7: Gráfico del error de seguimiento de dirección

4.2 Pruebas de campo

Para realizar las pruebas reales es necesario modificar el esquema de control, en este caso se debe tener en cuenta que el programa se debe comunicar con ROS y este a su vez con el Husky, por tanto se desarrolla un nuevo flujo de datos representado por el esquema de la figura 4.8, para ejemplificar el flujo de datos del robot.

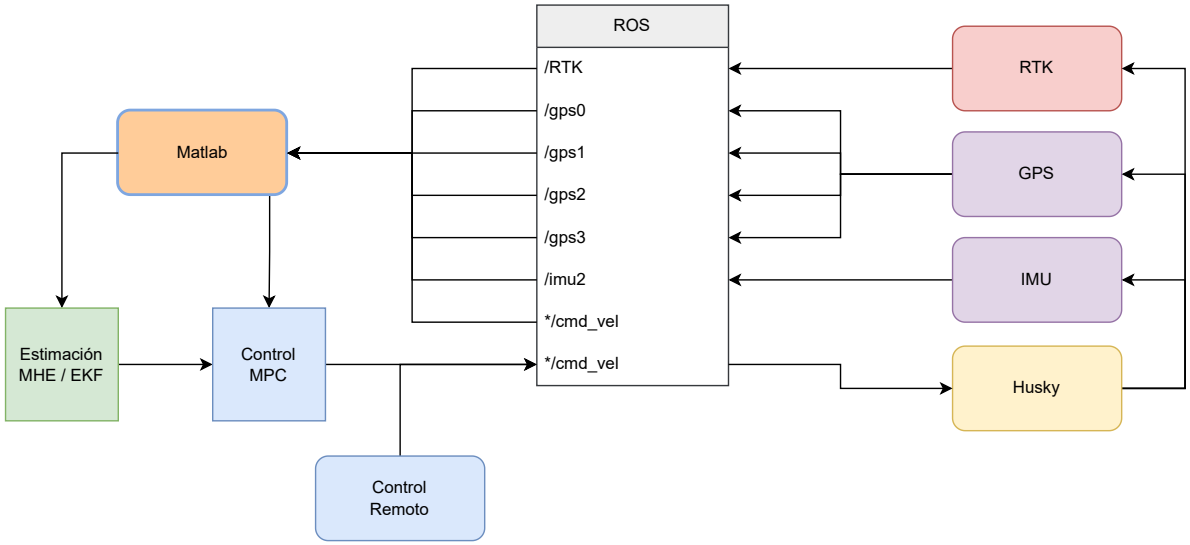


Figura 4.8: Esquema de interconexiones para pruebas de campo.

El entorno de pruebas designado para hacer las rutas del robot fue la azotea del edificio del centro de investigación AC3E resaltado en la figura 4.9, ubicado en Gral. Bari 699, Valparaíso, Chile. Se realizaron las pruebas del robot real como muestra la imagen de la figura 4.10, siguiendo distintas rutas en las cuales se generaron archivos de ROS tipo bag, estos archivos permiten guardar todos los datos entregados por los sensores al computador y desde el computador al robot, acompañado ademas de una estampa de tiempo que permite verificar en que momento se recibe cada muestra.

Figura 4.9: Azotea edificio AC3E.

Figura 4.10: Husky en la azotea

De todas las trayectorias generadas en las pruebas de campo se eligieron dos situaciones en particular, representando situaciones diferentes que permiten comprobar el desempeño del algoritmo, que a continuación se exponen. En ambos casos se utiliza los mismos parámetros para ambos estimadores.

En MHE se tienen dos matrices pesos en las cuales se aplicó una metodología de punto de partida para posteriormente realizar prueba y error buscando el mejor ajuste del promedio del error de posición. Para el caso de la matriz W se tiene el punto de partida como una diagonal de unos, esto quiere decir que se asumió inicialmente que el Husky no induce ruido de proceso para un vector de estado χ, pero se obtuvo mejores resultados aumentando 10 veces valor correspondiente a θ, esto implicando el ruido que induce el robot en su dirección angular en radianes es mas bajo en magnitud que para la posición en metros:

$$\chi = \begin{bmatrix} \theta \\ x \\ y \end{bmatrix}$$

$$W = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para el caso de la matriz R se tiene la cantidad de estados x,y se multiplica por 4 ya que se tienen 4 GPS, con el vector de estado χ_{MHE}. La matriz R se elige con los valores 100 para la medición de

ángulo, 0.1 para los GPS de bajo rendimiento y 10 para los GPS de alto rendimiento, y en este caso no se obtuvieron mejores resultados que la suposición inicial.

$$\chi_{MHE} = \begin{bmatrix} \theta \\ x_0 \\ y_0 \\ x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \end{bmatrix}$$

$$R = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix}$$

Mientras que el EKF cuenta con los siguientes parámetros, que en este caso son matrices 3x3, dado que no permite agregar restricciones que permitan combinar los 4 GPS se realiza un promedio simple antes de realizar la estimación obteniéndose un vector de estados anteriormente visto χ, y es por esto que se cuenta con números mas pequeños que en las matrices anteriores, dado que EKF utiliza las covarianzas de ruidos gaussianos mientras que MHE, si los ruidos fueran gaussianos los valores de la diagonal serian $1/covarianzas$. Para la matriz Qk se comenzó en valores de 0.0001 y con prueba y error se encontraron los mejores valores.Por otro lado en la determinación de la matriz Rk utilizó una muestra de la covarianza entregada por el paquete de ROS Navsat promediadas para la posición, mientras que para dirección se utilizó la inversa del valor obtenido en MHE:

$$Qk = \begin{bmatrix} 0.005 & 0 & 0 \\ 0 & 0.005 & 0 \\ 0 & 0 & 0.005 \end{bmatrix}$$

$$Rk = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.053125 & 0 \\ 0 & 0 & 0.053125 \end{bmatrix}$$

El código completo se encuentra en el Anexo.

4.2.1 Ruta extensa

Primero se realiza una prueba con una ruta larga, en la que se pueda observar el comportamiento del estimador en una situación en la que el robot se desplazó una gran distancia. Esta ruta de encuentra descrita en al figura 4.12. Se encuentra una mejora promedio del 60% y del 88% en el mejor de los casos en la estimación al comparar MHE con EKF, por lo que en este caso MHE muestra ser mucho mejor.

Error promedio MHE: $0.8173[m]$

Error promedio EKF: $2.0474[m]$

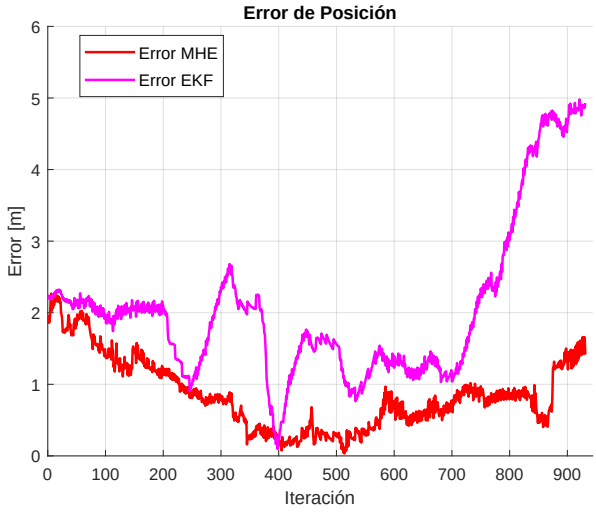


Figura 4.11: Errores respecto a la medición de control en ruta extensa

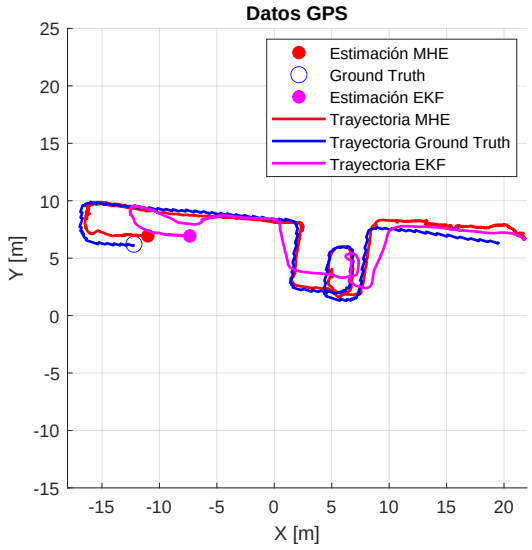


Figura 4.12: Comportamiento de la estimación en ruta extensa

4.2.2 Ruta reducida

También se analiza el comportamiento del algoritmo en el caso de un entorno reducido como muestra la figura 4.14. Ademas se tiene la comparación del filtro extendido de Kalman y el MHE en la figura 4.13. En este caso se tiene una mejora promedio en la estimación del 5.8%, lo cual no se considera significativo, concluyendo que en este caso ambos tuvieron un comportamiento similar, aunque MHE se muestra mas ruidoso en los gráficos.

Error promedio MHE: $1.6237[m]$

Error promedio EKF: $1.7239[m]$

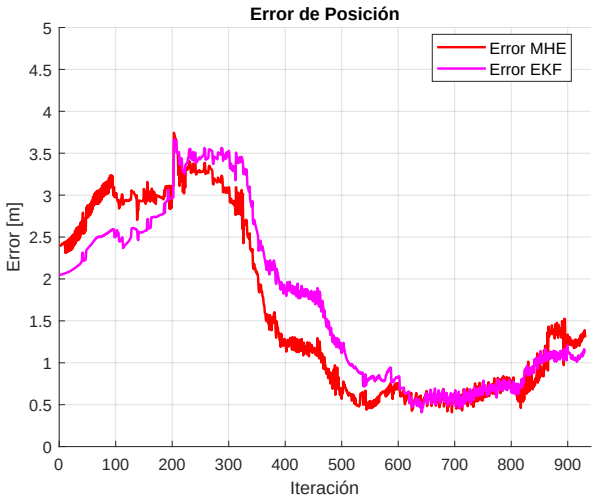


Figura 4.13: Comportamiento de la estimación de una ruta reducida

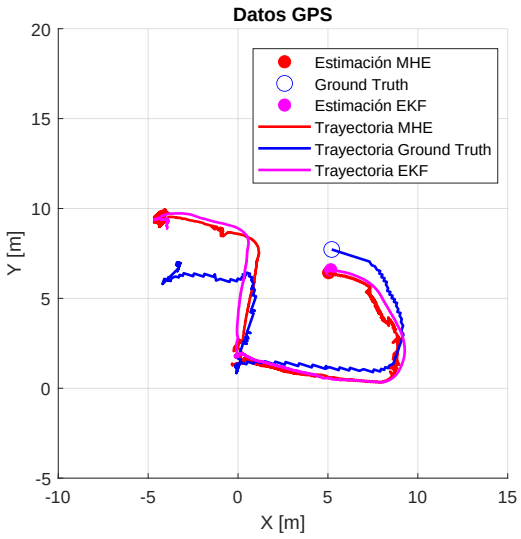


Figura 4.14: Comportamiento de la estimación de una ruta reducida

5

Conclusión

En este trabajo se evaluaron y compararon dos técnicas avanzadas de estimación para la fusión de datos provenientes de cuatro unidades GPS de diferente precisión: la Estimación de Horizonte Móvil (MHE) y el Filtro Extendido de Kalman (EKF). Se implementaron ambos algoritmos y se analizaron sus desempeños en entornos simulados y pruebas de campo con el robot Husky A200.

Los resultados en simulación mostraron que MHE logra una mejor estimación de la posición en comparación con EKF, especialmente en escenarios donde las condiciones iniciales presentan grandes desviaciones. Además, al integrarse con un controlador predictivo basado en MPC, se logró un seguimiento de trayectoria preciso de la trayectoria lemniscata propuesta a una velocidad de $1[m/s]$, con errores de posicionamiento controlados dentro de un margen de 0.5 metros.

Los resultados obtenidos a partir de las pruebas de campo realizadas muestran un mejor comportamiento del MHE que EKF. Esto se debe a que el error derivado de las condiciones iniciales es considerable y se mantiene durante el trayecto, lo que se traduce en una reducción significativa del error cuando las condiciones no-lineales son desfavorables para el EKF. En el escenario más extremo (véase la figura 4.11), el EKF presentó errores de hasta 4 metros, mientras que el MHE logró disminuirlos a apenas 0.5 metros, presentado una mejora de hasta el 88% en el mejor caso.

Para trayectorias de corta distancia, la diferencia entre ambos métodos resulta menos marcada, con una diferencia promedio del 5.8% que se considera un desempeño equivalente para ambos algoritmos, por lo que si se busca diminuir el error la mejor opción en general es usar MHE. Sin embargo, es importante resaltar que el MHE demanda un mayor consumo computacional, presentándose como una restricción para robots que operen a altas velocidades o en trayectorias con curvas muy cerradas propias de entornos reducidos. En este estudio se utilizó un Husky A200 con una velocidad máxima de $1[m/s]$ y un laptop con Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz con 16 GB de RAM.

Finalmente, el apoyo del AC3E y la orientación del profesor guía fueron fundamentales para llevar a cabo y concretar este proyecto con éxito.

Referencias

[1] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. doi: 10.1115/1.3662552.

[2] S. F. Schmidt. The kalman filter – its recognition and development for aerospace applications. *Journal of Guidance and Control*, 4(1):4–7, 1981. doi: 10.2514/3.19713.

[3] S. Kn"ufer and M. A. M"uller. Nonlinear full information and moving horizon estimation: Robust global asymptotic stability. *Automatica*, 150:110603, 2023.

[4] Daniel Etiemble. 45-year cpu evolution: one law and two equations, 2018. arXiv:1803.00254.

[5] Bruno Siciliano, Oussama Khatib, and Torsten Kröger. *Springer handbook of robotics*, volume 200. Springer, 2008.

[6] R. Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation. *IEEE Signal Processing Magazine*, 29(5):128–132, 2012.

[7] J. B. Rawlings, D. Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. 2017.

[8] C. V. Rao, J. B. Rawlings, and J. H. Lee. Constrained linear state estimation: a moving horizon approach. *Automatica*, 37(10):1619–1628, 2001.

[9] Clearpath Robotics. Husky a200 user manual, 2025. URL `https://docs.clearpathrobotics.com/docs/robots/outdoor_robots/husky/user_manual_husky`.

[10] u-blox. Neo-7 series, 2024. URL `https://www.u-blox.com/en/product/neo-7-series`.

[11] VectorNav. Vn-200 gnss/ins, 2024. URL `https://www.vectornav.com/products/detail/vn-200`.

[12] Navcom. Sf-3040 — gnss user guide, 2017. URL `https://www.waps.net.au/wp-content/uploads/2017/02/4-Navcom-SF-3040-User-Guide.pdf`.

[13] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Robotics*, 2009.

[14] The MathWorks, Inc. Matlab version: 9.13.0 (r2022b), 2022.

[15] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. Casadi — a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1): 1–36, 2019.

[16] L. T. Biegler and V. M. Zavala. Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers and Chemical Engineering*, 33(3): 575–582, 2009.

Anexo A

Códigos

A.1 Simulación

Este es el código utilizado para simular el movimiento del robot en matlab.

```
1
2  clearvars
3  close all
4  clc
5
6  addpath(fullfile(pwd, 'casadi-3.6.5-windows64-matlab2018b'))
7  import casadi.*
8
9  Ts = 0.1;
10 sampling_factor=1000;
11 sim_Ts = Ts/sampling_factor;
12
13 gps1_noise_factor= 0.1*1;
14 gps2_noise_factor= 0.1*2;
15 gps3_noise_factor= 0.1*4;
16 gps4_noise_factor= 0.1*8;
17
18 gps_noise_factor = sqrt((gps1_noise_factor^2+gps2_noise_factor^2+gps3_noise_factor^2+gps4_noise_factor^2)/16);
19 gps_noise_factor_variance = gps_noise_factor^2;
20
21 gyro_noise_factor = 2 * pi * 0.005;
22 gyro_noise_factor_variance = gyro_noise_factor^2;
23
24 w = 0.555;
25 l = 0.544;
26
27 wg = 0.6;
28 lg = 0.6;
29
30 vel_max = 4;
31
32 wheel_radius = 0.1;
33
34 nq = 3;
35 nu = 2;
36 nr = 3;
37
38 Nc = 30;
39
40 max_angular_velocity = vel_max / wheel_radius;
41 wheel_vel = max_angular_velocity/2;
42
43 U_lb = [-wheel_vel; -wheel_vel];
44 U_ub = [wheel_vel; wheel_vel];
45
```

```
dU_lb = U_lb * 0.1;
dU_ub = U_ub * 0.1;

optiMPC = casadi.Opti();

qMPC = optiMPC.variable(nq, Nc + 1);
u = optiMPC.variable(nu, Nc);
qref = optiMPC.parameter(nr);
qinit = optiMPC.parameter(nq);
u_last = optiMPC.parameter(nu);

Wmpc = diag([1, 1, 2] * 2);
Rmpc = diag([0.08, 0.08]);

Jmpc = 0;
for k = 1:Nc + 1
    error = qMPC(:, k) - qref;
    for i = 1:6
        error(3) = if_else(error(3) > pi, error(3) - 2 * pi, error(3));
        error(3) = if_else(error(3) < -pi, error(3) + 2 * pi, error(3));
    end

    Jmpc = Jmpc + (error).' * Wmpc * (error);
    if k < Nc + 1
        optiMPC.subject_to(qMPC(:, k + 1) == F(qMPC(:, k), u(:, k), Ts, w, zeros(3, 1)));

        if k ~= 1
            optiMPC.subject_to((dU_lb <= (u(:, k) - u(:, k - 1)) / Ts) <= dU_ub);
        else
            optiMPC.subject_to((dU_lb <= (u(:, k) - u_last) / Ts) <= dU_ub);
        end
        optiMPC.subject_to((U_lb <= u(:, k)) <= U_ub);
        Jmpc = Jmpc + u(:, k).' * Rmpc * u(:, k);
    end
end

optiMPC.subject_to(u(1, :) * 0.5 + u(2, :) * 0.5 >= 0);

optiMPC.minimize(sum(sum(u.^2)));

optiMPC.minimize(Jmpc);

optiMPC.subject_to(qMPC(:, 1) == qinit);

p_optsMPC = struct('expand', true);
s_optsMPC = struct('sb', 'yes', 'print_level', 0, 'warm_start_init_point', 'yes');
optiMPC.solver('ipopt', p_optsMPC, s_optsMPC);

Nh = 20;
optiMHE = casadi.Opti();

qMHE = optiMHE.variable(nq, Nh + 1);
wMHE = optiMHE.variable(nq, Nh);
vMHE = optiMHE.variable(nq, Nh + 1);

alphaMHE = optiMHE.variable(4, Nh+1);

y_meas = optiMHE.parameter(4 * 2 + 1, Nh + 1);
y_meas_convx = optiMHE.variable(3, Nh + 1);

uPast = optiMHE.parameter(nu, Nh);
X = optiMHE.variable(nq);
xBar = optiMHE.parameter(nq);
Pmhe = optiMHE.parameter(nq,nq);
JmhePast = optiMHE.parameter(1);

Rmhe = diag([1/gps_noise_factor_variance, 1/gps_noise_factor_variance, 1/gyro_noise_factor_variance])*0.15;
Qmhe = diag([1, 1, 1/(2*pi)] * 800);

Jmhe = X.' * Pmhe * X + JmhePast;
optiMHE.subject_to(X == qMHE(:, 1) - xBar);

for k = 1:Nh
    Jmhe = Jmhe + vMHE(:, k).' * Rmhe * vMHE(:, k) + wMHE(:, k).' * Qmhe * wMHE(:, k);
```

```
120
121     optiMHE.subject_to(qMHE(:, k + 1) == F(qMHE(:, k), uPast(:, k), Ts, w, wMHE(:, k)));
122
123     R = [cos(y_meas(4 * 2 + 1, k)), -sin(y_meas(4 * 2 + 1, k));
124          sin(y_meas(4 * 2 + 1, k)),  cos(y_meas(4 * 2 + 1, k))];
125
126     optiMHE.subject_to(sum(alphaMHE(:,k))==1);
127     optiMHE.subject_to((0<=alphaMHE(:,k))<=1);
128
129     y_meas_convx (:,k)= [(y_meas(1:2, k) - R * 0.5 * [lg; -wg]) * alphaMHE(1, k) + ...
130                          (y_meas(3:4, k) - R * 0.5 * [lg; wg]) * alphaMHE(2, k) + ...
131                          (y_meas(5:6, k) - R * 0.5 * [-lg; wg]) * alphaMHE(3, k) + ...
132                          (y_meas(7:8, k) - R * 0.5 * [-lg; -wg]) * alphaMHE(4, k); y_meas(4 * 2 + 1, k)];
133
134     optiMHE.subject_to(qMHE(:, k) + vMHE(:, k) == (y_meas_convx (:,k)));
135 end
136
137 R = [cos(y_meas(4 * 2 + 1, Nh + 1)), -sin(y_meas(4 * 2 + 1, Nh + 1));
138      sin(y_meas(4 * 2 + 1, Nh + 1)),  cos(y_meas(4 * 2 + 1, Nh + 1))];
139
140 optiMHE.subject_to(sum(alphaMHE(:,Nh+1))==1);
141 optiMHE.subject_to((0<=alphaMHE(:,Nh+1))<=1);
142
143 y_meas_convx(:,k) = [(y_meas(1:2, Nh + 1) - R * 0.5 * [lg; -wg]) * alphaMHE(1, Nh + 1) + ...
144                 (y_meas(3:4, Nh + 1) - R * 0.5 * [lg; wg]) * alphaMHE(2, Nh + 1) + ...
145                 (y_meas(5:6, Nh + 1) - R * 0.5 * [-lg; wg]) * alphaMHE(3, Nh + 1) + ...
146                 (y_meas(7:8, Nh + 1) - R * 0.5 * [-lg; -wg]) * alphaMHE(4, Nh + 1); y_meas(4 * 2 + 1, Nh + 1)];
147
148 optiMHE.subject_to(qMHE(:, Nh + 1) + vMHE(:, Nh + 1) == y_meas_convx(:,Nh+1) );
149
150 Jmhe = Jmhe + vMHE(:, Nh + 1).' * Rmhe * vMHE(:, Nh + 1);
151
152 optiMHE.minimize(Jmhe);
153
154 optiMHE.subject_to((-gps_noise_factor*3 <= wMHE) <= gps_noise_factor*3 );
155 optiMHE.subject_to((-gps_noise_factor*3 <= vMHE) <= gps_noise_factor*3 );
156
157 p_optsMHE = struct('expand', true);
158 s_optsMHE = struct('sb', 'yes', 'print_level', 0, 'gamma_theta', 1e-2, 'jacobian_approximation', 'exact', ...
159 'fast_step_computation', 'yes', 'warm_start_init_point', 'yes');
160 optiMHE.solver('ipopt', p_optsMHE, s_optsMHE);
161
162 cicles = 0.95;
163
164 ref_samples = 0: sim_Ts :2 * pi * cicles;
165
166 scale = 6;
167
168 x_lim = scale;
169 y_lim = scale;
170
171 x_path = scale * cos(ref_samples) ./ (1 + sin(ref_samples).^2);
172 y_path = scale * sin(ref_samples) .* cos(ref_samples) ./ (1 + sin(ref_samples).^2);
173 thetaref = unwrap(atan2(diff(y_path), diff(x_path)));
174
175 qr = [x_path; y_path; [thetaref, thetaref(end)]];
176
177 distance = sqrt(diff(x_path).^2 + diff(y_path).^2);
178 total_distance = sum(distance);
179
180 vref = 1;
181
182 total_time = total_distance / vref;
183
184 t = 0: Ts : total_time;
185
186 num_samples = length(t);
187
188 downsampling_factor = floor(size(qr, 2) / num_samples);
189 qr_downsampled = qr(:, 1:downsampling_factor:end);
190
191 if size(qr_downsampled, 2) < num_samples
192     qr_downsampled = [qr_downsampled, qr(:, end)];
193 end
```

```
194
195 qr = qr_downsampled;
196
197 num_samples = size(qr, 2);
198 t = linspace(0, total_time, num_samples);
199 optiMPC.set_value(u_last, [0; 0]);
200
201 q0 = [4, -1, 1];
202 R = [cos(q0(3)), -sin(q0(3)); sin(q0(3)), cos(q0(3))];
203 gps0_ = R * 0.5 * [lg, lg, -lg, -lg; -wg, wg, wg, -wg] + repmat(q0(1:2)', 1, 4);
204 gps0Bar = [gps0_(:) + [gps1_noise_factor .* randn(2,1); gps2_noise_factor .* randn(2,1); gps3_noise_factor .* randn(2,1); ...
205 gps4_noise_factor .* randn(2,1)]; q0(3) + gyro_noise_factor .* randn(1)]+scale*[repmat([0.2;0.2],4,1);pi*0.1]*0;
206 gps0Bar_mean = [mean(gps0Bar(1:2:end - 1)); mean(gps0Bar(2:2:end - 1)); gps0Bar(end)];
207
208 Q = zeros(nq, length(t));
209 Q(:, 1) = q0;
210 Qr = zeros(3, length(t));
211 Qe = zeros(3, length(t));
212 Qestimated = zeros(nq, length(t));
213 Qmean = zeros(nq, length(t));
214 u_mpc = zeros(nu, length(t));
215 u_kc = zeros(nu, length(t));
216
217 optiMPC.set_value(u_last, [0; 0]);
218
219 optiMHE.set_value(uPast, repmat([0; 0], 1, Nh));
220 optiMHE.set_value(xBar, gps0Bar_mean(:));
221 optiMHE.set_value(JmhePast, 0);
222 optiMHE.set_value(Pmhe, diag([1/gps_noise_factor_variance, 1/gps_noise_factor_variance, 1/gyro_noise_factor_variance]));
223 optiMHE.set_value(y_meas, repmat(gps0Bar(:), 1, Nh + 1));
224 optiMHE.set_value(uPast, repmat([0; 0], 1, Nh));
225 optiMHE.set_initial(alphaMHE, repmat([0.25; 0.25; 0.25; 0.25], 1, Nh + 1));
226 optiMHE.set_initial(qMHE, repmat(gps0Bar_mean(:),1,Nh+1));
227 solutionMHE = optiMHE.solve();
228
229 elapsed_time = zeros(1, length(qr));
230
231 figure(1);
232 hold on;
233 grid on;
234 title('Seguimiento de trayectoria');
235 xlim([-x_lim * 1.1, x_lim * 1.1]);
236 ylim([-y_lim * 1.1, y_lim * 1.1]);
237 xlabel('X [m]');
238 ylabel('Y [m]');
239 daspect([1 1 1])
240
241 h1 = plot(NaN, NaN, 'k', 'DisplayName', 'Reference Trajectory', 'LineWidth', 1.5);
242 h2 = plot(NaN, NaN, 'b', 'DisplayName', 'Actual Position', 'LineWidth', 1.5);
243 h3 = plot(NaN, NaN, 'r--', 'DisplayName', 'Estimated Position', 'LineWidth', 1.5);
244 h5 = plot(NaN, NaN, 'm.', 'DisplayName', 'Kalman Filter', 'LineWidth', 1.5);
245 h_triangle = [];
246
247 legend([h1, h2, h3, h5]);
248
249 Qk = 0.00001 * eye(3);
250 Rk = diag([gps_noise_factor_variance, gps_noise_factor_variance, gyro_noise_factor_variance]);
251
252 x_kalman = gps0Bar_mean;
253 P_kalman = Rk;
254
255 Qkalman = zeros(size(Q));
256
257 Qmean(:, 1)=gps0Bar_mean(:);
258 last_measurement=gps0Bar(:);
259
260 for i = 1:length(qr)
261
262     if i ~= 1
263     last_gps = repmat(Q(1:2, i), 1, 4) + [cos(Q(3, i)), -sin(Q(3, i)); ...
264     sin(Q(3, i)), cos(Q(3, i))] * 0.5 * [lg, lg, -lg, -lg; -wg, wg, wg, -wg];
265     last_measurement = [last_gps(:) + [gps1_noise_factor .* randn(2,1); gps2_noise_factor .* randn(2,1);...
266     gps3_noise_factor .* randn(2,1); gps4_noise_factor .* randn(2,1)]; Q(3, i) + gyro_noise_factor .* randn(1)];
267     Qmean(:, i) = [mean(last_measurement(1:2:end - 1)); mean(last_measurement(2:2:end - 1)); last_measurement(end)];
```

```
    end

    tic;

    x_pred = F(x_kalman, u_mpc(:, i), Ts, w, zeros(3, 1));

    theta = x_kalman(3);
    v = (u_mpc(1, i) + u_mpc(2, i)) / 2;

    F_jacobian = [
        1, 0, -Ts * v * sin(theta);
        0, 1,  Ts * v * cos(theta);
        0, 0, 1
    ];

    P_pred = F_jacobian * P_kalman * F_jacobian' + Qk;

    H_jacobian = eye(3);

    y_tilde = Qmean(:, i) - x_pred;

    S = H_jacobian * P_pred * H_jacobian' + Rk;
    K = P_pred * H_jacobian' / S;

    x_kalman = x_pred + K * y_tilde;
    P_kalman = (eye(3) - K * H_jacobian) * P_pred;

    Qkalman(:, i) = x_kalman;

    P_kalman_inv=inv(P_kalman);
    optiMHE.set_value(Pmhe, P_kalman_inv);

    qTrajEstimated = solutionMHE.value(qMHE);
    yTrajEstimated = solutionMHE.value(y_meas);
    uTrajEstimated = solutionMHE.value(uPast);

    optiMHE.set_value(xBar, qTrajEstimated(:, 2));

    optiMHE.set_value(y_meas, [yTrajEstimated(:, 2:end), last_measurement]);
    optiMHE.set_value(uPast, [uTrajEstimated(:, 2:end), u_mpc(:, i)]);
    solutionMHE = optiMHE.solve();
    optiMHE.set_initial(solutionMHE.value_variables);
    optiMHE.set_value(JmhePast, solutionMHE.value(Jmhe));
    Qaux = solutionMHE.value(qMHE);
    Qestimated(:, i) = Qaux(:, end);

    if i < length(qr)
        optiMPC.set_value(qref, qr(:, i));
        optiMPC.set_value(qinit, Qestimated(:, i));
        try
            solutionMPC = optiMPC.solve();
            optiMPC.set_initial(solutionMPC.value_variables());
            u_mpc(:, i + 1) = solutionMPC.value(u(:, 1));
            optiMPC.set_value(u_last, u_mpc(:, i + 1));
        catch
            disp('MPC Solver failed');
            u_mpc(:, i + 1) = u_mpc(:, i);
        end

        elapsed_time(i) = toc;

        if i == round(length(qr) / 3)
        end
        Q(:, i + 1) = F(Q(:, i), u_mpc(:, i), Ts, w, zeros(3, 1));
    end

    disp('Max elapsed time');
    disp((elapsed_time(i)));

    cla;
    plot(qr(1, 1:i), qr(2, 1:i), 'k', 'DisplayName', 'Reference Trajectory', 'LineWidth', 1.5);
    plot(Q(1, 1:i), Q(2, 1:i), 'b', 'DisplayName', 'Actual Position', 'LineWidth', 1.5);
    plot(Qestimated(1, 1:i), Qestimated(2, 1:i), 'r--', 'DisplayName', 'MHE Position', 'LineWidth', 1.5);
    plot(Qkalman(1, 1:i), Qkalman(2, 1:i), 'm.', 'DisplayName', 'EKF Position', 'LineWidth', 1.5);
```

```
    plot(last_measurement(1:2:end - 1), last_measurement(2:2:end - 1), 'ro', ...
    'DisplayName', 'GPS noisy measurements');
    plot(solutionMHE.value(y_meas_convx(1,end-1)), ...
    solutionMHE.value(y_meas_convx(2,end-1)), 'r*', 'DisplayName', 'GPS estimated measurements');

    if ~isempty(h_triangle)
        delete(h_triangle);
    end
    head_width = 0.3;
    head_length = head_width;
    theta = Q(3, i);

    x_triangle = [0, -head_length, -head_length];
    y_triangle = [0, head_width / 2, -head_width / 2];

    R = [cos(theta), -sin(theta); sin(theta), cos(theta)];
    triangle = R * [x_triangle + head_length * 0.5; y_triangle]*scale/4;
    h_triangle = fill(Q(1, i) + triangle(1, :), Q(2, i) + triangle(2, :), 'b', 'EdgeColor', 'none', 'HandleVisibility', 'off');

    drawnow;
end

figure(2);
hold on;
grid on;
plot(t, u_mpc(1, :), 'Color', [0.3010, 0.7450, 0.9330], 'DisplayName', 'Entrada de control MPC 1 VR');
plot(t, u_mpc(2, :), 'Color', [0.8500, 0.3250, 0.0980], 'DisplayName', 'Entrada de control MPC 2 VL');
legend('show');
title('Entradas de Control');
xlabel('Tiempo [s]');
ylabel('Entrada de Control');

figure(3);
subplot(1, 2, 1);
hold on;
grid on;
plot(t(1:end-1), sqrt((Qestimated(1, 1:end-1) - Q(1, 1:end-1)).^2 + ...
(Qestimated(2, 1:end-1) - Q(2, 1:end-1)).^2), 'r', 'DisplayName', 'Error de posición MHE');
plot(t(1:end-1), sqrt((Qkalman(1, 1:end-1) - Q(1, 1:end-1)).^2 + ...
(Qkalman(2, 1:end-1) - Q(2, 1:end-1)).^2), 'm', 'DisplayName', 'Error de posición EKF');
legend('show');
title('Error de Posición');
xlabel('Tiempo [s]');
ylabel('Error de Posición');

subplot(1, 2, 2);
hold on;
grid on;
plot(t(1:end-1), abs(Qestimated(3, 1:end-1) - Q(3, 1:end-1)), 'r', 'DisplayName', 'Error de dirección MHE');
plot(t(1:end-1), abs(Qkalman(3, 1:end-1) - Q(3, 1:end-1)), 'm', 'DisplayName', 'Error de dirección EKF');
legend('show');
title('Error en Dirección');
xlabel('Tiempo [s]');
ylabel('Error en Dirección');
ylim([0, 0.25]);

figure(4);
hold on;
grid on;
plot(elapsed_time, 'b', 'DisplayName', 'Tiempo por iteración');
legend('show');
title('Tiempo de Ejecución por Iteración');
xlabel('Iteración');
ylabel('Tiempo de Ejecución (segundos)');

figure(5);
subplot(1, 2, 1);
hold on;
grid on;
plot(sqrt((Q(1, :) - qr(1, :)).^2 + (Q(2, :) - qr(2, :)).^2), 'b', 'DisplayName', 'Error de posición Q');
legend('show');
title('Error de Posición vs Referencia');
xlabel('Tiempo [s]');
```

```
ylabel('Error de Posición');

subplot(1, 2, 2);
hold on;
grid on;
plot(Q(3, :) - qr(3, :), 'b', 'DisplayName', 'Error de dirección Q');
legend('show');
title('Error en Dirección vs Referencia');
xlabel('Tiempo [s]');
ylabel('Error en Dirección');

disp('Diferencia media del error en posición EKF-MHE:');
disp(sqrt(mean((Qkalman(1, 1:end-1) - Q(1, 1:end-1)).^2 + (Qkalman(2, 1:end-1) - ...
Q(2, 1:end-1)).^2))-mean(sqrt((Qestimated(1, 1:end-1) - Q(1, 1:end-1)).^2 + (Qestimated(2, 1:end-1) - Q(2, 1:end-1)).^2)));

disp('Diferencia media del error en dirección EKF-MHE:');
disp(mean(abs(Qkalman(3, 1:end-1) - Q(3, 1:end-1)))-mean(abs(Qestimated(3, 1:end-1) - Q(3, 1:end-1))));

disp('Error medio de estimación MHE:');
disp(mean(sqrt((Q(1, :) - Qestimated(1, :)).^2 + (Q(2, :) - Qestimated(2, :)).^2)));

function Fk = F(q, u, Ts, w, noise)
    Fk = q + Ts * [cos(q(3)), 0;
                   sin(q(3)), 0;
                   0,         1] * [1/2, 1/2; 1/w, -1/w] * u + noise;
end
```

Código A.1: Implementación de simulación

A.2 Rosbag

Este es el código utilizado para extrae los datos adquiridos por ros y procesar la información.

```
clear all
close all
clc

import casadi.*
% General Parameters:
% Convert Latitude and Longitude to meters
origin                      = [-33.034115, -71.592205];
wf                          = 0.43;
wr                          = 0.32;
l                           = 0.475;

bag                     = rosbag('2025-02-27-17-45-52.bag');
gps0_data               = select(bag, 'Topic', '/gps0');
gps1_data               = select(bag, 'Topic', '/gps1');
gps2_data               = select(bag, 'Topic', '/gps2');
gps3_data               = select(bag, 'Topic', '/gps3');
rtk_data                = select(bag, 'Topic', '/rtk');

husky_cmdVel            = select(bag, 'Topic','/husky_velocity_controller/cmd_vel');
husky_odomVel           = select(bag, 'Topic','/husky_velocity_controller/odom');
husky_prmsVel           = select(bag, 'Topic','/husky_velocity_controller/parameter_descriptions');

imu2                    = select(bag, 'Topic', '/imu2');
imu3                    = select(bag, 'Topic', '/imu3');

vecINS                  = select(bag, 'Topic', '/vectornav/INS');
vecOdom                 = select(bag, 'Topic', '/vectornav/Odom');

for i=1:1
    vecOdomData = readMessages(vecOdom,i);
    linVelZ     = vecOdomData{1,1}.Twist.Twist.Linear.Z;
    linVel      = sqrt(vecOdomData{1,1}.Twist.Twist.Linear.X^2+...
                    vecOdomData{1,1}.Twist.Twist.Linear.Y^2)*sign(vecOdomData{1,1}.Twist.Twist.Linear.X);
    angVel      = vecOdomData{1,1}.Twist.Twist.Angular.Z;
```

```
    huskyOdomVelData = readMessages(husky_odomVel,i);
    eulAngle    = quat2eul([huskyOdomVelData{1,1}.Pose.Pose.Orientation.X, ...
    huskyOdomVelData{1,1}.Pose.Pose.Orientation.Y,huskyOdomVelData{1,1}.Pose.Pose.Orientation.Z,huskyOdomVelData{1,1}.Pose.Pose.Orientation.W]);
    huskyAtt    = eulAngle(3);

    imu2Data    = readMessages(imu2,i);
    eulAngle    = quat2eul([imu2Data{1,1}.Orientation.X,imu2Data{1,1}.Orientation.Y, ...
    imu2Data{1,1}.Orientation.Z,imu2Data{1,1}.Orientation.W]);
    imu2Att     = eulAngle(3);

    imu3Data    = readMessages(imu3,i);
    eulAngle    = quat2eul([imu3Data{1,1}.Orientation.X,imu3Data{1,1}.Orientation.Y, ...
    imu3Data{1,1}.Orientation.Z,imu3Data{1,1}.Orientation.W]);
    imu3Att     = eulAngle(3);
end

% Compute rotation matrix to navigate in my local reference frame
sdpvar a11 a12 a21 a22;
sdpvar x1bar y1bar x2bar y2bar;
% AZOTEA AC3E
lat1                        = -33.034213;
long1                       = -71.592168;
[x1, y1]                    = latlon2xy(lat1, long1, origin(1), origin(2));
x1                          = x1*1000;
y1                          = y1*1000;
x                           = norm([x1 y1]);
%
lat2                        = -33.034088;
long2                       = -71.59211333;
[x2, y2]                    = latlon2xy(lat2, long2, origin(1), origin(2));
x2                          = x2*1000;
y2                          = y2*1000;
y                           = norm([x2 y2]);

A                           = [a11 a12; a21 a22];
v                           = [x1; y1; x2; y2];
b                           = [x1bar; y1bar; x2bar; y2bar];
Constraints                 = [[A zeros(2); zeros(2) A]*v - b == zeros(4,1); x1bar*x2bar + y1bar*y2bar == 0];

obj                         = (x1bar - x)^2 + (y2bar - y)^2;
optimize(Constraints, obj);
mtxRot                      = value(A);


msg0                        = readMessages(gps0_data, 1);
msg1                        = readMessages(gps1_data, 1);
msg2                        = readMessages(gps2_data, 1);
msg3                        = readMessages(gps3_data, 1);
msgRTK                      = readMessages(rtk_data, 1);

[offset_x, offset_y]        = latlon2xy(msgRTK{1,1}.Latitude, msgRTK{1,1}.Longitude, origin(1), origin(2));
rtx_xy                      = [offset_x; offset_y] * 1000;
rtx_xy                      = mtxRot * rtx_xy;
%
[gps0_x, gps0_y]            = latlon2xy(msg0{1,1}.Latitude, msg0{1,1}.Longitude, origin(1), origin(2));
gps0_xy                     = [gps0_x; gps0_y] .* 1000;
gps0_xy                     = mtxRot * gps0_xy;
offset_x0                   = gps0_xy(1)-rtx_xy(1) - l/2;
offset_y0                   = gps0_xy(2)-rtx_xy(2) - wf/2;
cov_gps0_x                  = msg0{1,1}.PositionCovariance(1);
cov_gps0_y                  = msg0{1,1}.PositionCovariance(5);
%
[gps1_x, gps1_y]            = latlon2xy(msg1{1,1}.Latitude, msg1{1,1}.Longitude, origin(1), origin(2));
gps1_xy                     = [gps1_x; gps1_y] .* 1000;
gps1_xy                     = mtxRot * gps1_xy;
offset_x1                   = gps1_xy(1)-rtx_xy(1) - l/2;
offset_y1                   = gps1_xy(2)-rtx_xy(2) + wf/2;
cov_gps1_x                  = msg1{1,1}.PositionCovariance(1);
cov_gps1_y                  = msg1{1,1}.PositionCovariance(5);

%
[gps2_x, gps2_y]            = latlon2xy(msg2{1,1}.Latitude, msg2{1,1}.Longitude, origin(1), origin(2));
gps2_xy                     = [gps2_x; gps2_y] .* 1000;
```

```
gps2_xy                     = mtxRot * gps2_xy;
offset_x2                   = gps2_xy(1)-rtx_xy(1) - -l/2;
offset_y2                   = gps2_xy(2)-rtx_xy(2) - wf/2;
cov_gps2_x                  = msg2{1,1}.PositionCovariance(1);
cov_gps2_y                  = msg2{1,1}.PositionCovariance(5);


[gps3_x, gps3_y]            = latlon2xy(msg3{1,1}.Latitude, msg3{1,1}.Longitude, origin(1), origin(2));
gps3_xy                     = [gps3_x; gps3_y] .* 1000;
gps3_xy                     = mtxRot * gps3_xy;
offset_x3                   = gps3_xy(1)-rtx_xy(1) + l/2;
offset_y3                   = gps3_xy(2)-rtx_xy(2) + wf/2;
cov_gps3_x                  = msg3{1,1}.PositionCovariance(1);
cov_gps3_y                  = msg3{1,1}.PositionCovariance(5);


% Kinematic model
nq                          = 3;
nu                          = 2;
ny4gps                      = 1+2*4;
Ts                          = 0.2;
q                           = casadi.MX.sym('q',nq);
u                           = casadi.MX.sym('u',nu);
f_rhs                       = [     u(1);...
                               cos(q(1)) * u(2);...
                               sin(q(1)) * u(2) ];

f                           = casadi.Function('f_rhs', {q,u}, {f_rhs});
opts                        = struct('main',true,'mex',true);
f.generate('f.c',opts)
mex f.c -largeArrayDims;
% Output of the system
h_rhs                       = q;
h                           = casadi.Function('h', {q}, {h_rhs});
% compute jacobians
jac_fx                      = casadi.Function('Func_A', {q, u}, {f_rhs.jacobian(q)}, {'x', 'u'}, {'A'});
jac_fu                      = casadi.Function('Func_B', {q, u}, {f_rhs.jacobian(u)}, {'x', 'u'}, {'B'});
jac_hx                      = casadi.Function('Func_C', {q}, {h_rhs.jacobian(q)}, {'x'}, {'C'});
% Integrator
k1                          = f(q, u);
k2                          = f(q + Ts / 2 * k1, u);
k3                          = f(q + Ts / 2 * k2, u);
k4                          = f(q + Ts * k3, u);
x_rk4                       = q + Ts / 6 * (k1 + 2 * k2 + 2 * k3 + k4);
FNt                         = casadi.Function('FNt', {q, u}, {x_rk4});
jac_FNt                     = casadi.Function('Func_D', {q, u}, {x_rk4.jacobian(q)}, {'x', 'u'}, {'D'});



% Initial orientation
imu2Data                    = readMessages(imu3,1);
eulAngle                    = quat2eul([imu2Data{1,1}.Orientation.X,imu2Data{1,1}.Orientation.Y, ...
imu2Data{1,1}.Orientation.Z,imu2Data{1,1}.Orientation.W]);
attInit                     = eulAngle(3);

% Init MHE with 4 GPSs
dims                        = {};
dims.nq                     = nq;
dims.nu                     = nu;
dims.ny                     = ny4gps;
boxConst                    = [];
Ne                          = 15;
Nt                          = 0;

mhe4gps                     = mheMemNic(Ne,Nt,FNt,h,dims,boxConst,Ts);

msg0                        = readMessages(gps0_data, 1);
[gps0_x, gps0_y]            = latlon2xy(msg0{1,1}.Latitude, msg0{1,1}.Longitude, origin(1), origin(2));
gps0_x                      = gps0_x*1000;
gps0_y                      = gps0_y*1000;
msg1                        = readMessages(gps1_data, 1);
[gps1_x, gps1_y]            = latlon2xy(msg1{1,1}.Latitude, msg1{1,1}.Longitude, origin(1), origin(2));
gps1_x                      = gps1_x*1000;
gps1_y                      = gps1_y*1000;
```

```
msg2                        = readMessages(gps2_data, 1);
[gps2_x, gps2_y]            = latlon2xy(msg2{1,1}.Latitude, msg2{1,1}.Longitude, origin(1), origin(2));
gps2_x                      = gps2_x*1000;
gps2_y                      = gps2_y*1000;
msg3                        = readMessages(gps3_data, 1);
[gps3_x, gps3_y]            = latlon2xy(msg3{1,1}.Latitude, msg3{1,1}.Longitude, origin(1), origin(2));
gps3_x                      = gps3_x*1000;
gps3_y                      = gps3_y*1000;

msgRTK                      = readMessages(rtk_data, 1);
[gpsRTK_x, gpsRTK_y]        = latlon2xy(msgRTK{1,1}.Latitude, msgRTK{1,1}.Longitude, origin(1), origin(2));
gpsRTK_x                    = gpsRTK_x*1000;
gpsRTK_y                    = gpsRTK_y*1000;
gpsRTK                      = mtxRot * [gpsRTK_x; gpsRTK_y];


x0bar                       = [imu3Att(1) - attInit; gpsRTK(1); gpsRTK(2)];
set_x0bar(mhe4gps,x0bar);
setJacobians(mhe4gps,jac_fx,jac_fu,jac_hx);
set_cPrm(mhe4gps,0.15);
set_alpha(mhe4gps,1);
setMtxW(mhe4gps,diag([10 1 1]));
setMtxR(mhe4gps,diag([100 0.1 0.1 0.1 0.1 10 10 10 10]));
setPrmsGPSpos(mhe4gps,[wf, wr, l]);

for i=1:Ne
    updateMeasurement(mhe4gps, [imu3Att(1); gps0_x-offset_x0; gps0_y-offset_y0;...
        gps1_x-offset_x1; gps1_y-offset_y1; gps2_x-offset_x2; gps2_y-offset_y2; gps3_x-offset_x3; gps3_y-offset_y3] );
    updateInput(mhe4gps,zeros(nu,1));
end
updateMeasurement(mhe4gps, [imu3Att(1); gps0_x-offset_x0; gps0_y-offset_y0;...
        gps1_x-offset_x1; gps1_y-offset_y1; gps2_x-offset_x2; gps2_y-offset_y2; gps3_x-offset_x3; gps3_y-offset_y3] );
alignFactor4gps             = [0; 2.8; 0.096];


% Init EKF

Qk                          = diag([0.0001 0.0001 0.0001])*50;
Rk                          = diag([0.01, 0.053125, 0.053125]);

efkgps                      = ekfMemNic(Ts, Qk, Rk, nq, ny4gps);
set_x0bar(efkgps, x0bar);

vecOdomData = readMessages(vecOdom, i);
linVelX = vecOdomData{1,1}.Twist.Twist.Linear.X;
linVelY = vecOdomData{1,1}.Twist.Twist.Linear.Y;
huskyLinVel = sqrt(linVelX^2 + linVelY^2) * sign(linVelX);
huskyAngVel = vecOdomData{1,1}.Twist.Twist.Angular.Z;

predict(efkgps, FNt, jac_FNt, [huskyAngVel,huskyLinVel]);
update(efkgps, [imu3Att(1); mean([gps0_x-offset_x0;gps1_x-offset_x1;gps2_x-offset_x2;gps3_x-offset_x3]); ...
     mean([gps0_y-offset_y0;gps1_y-offset_y1;  gps2_y-offset_y2;  gps3_y-offset_y3])], jac_hx );

alignFactorefk              = alignFactor4gps;

% Init some variables
attitudeOld                 = 0;
correctFactor               = 0;
estimatedPos4gps            = [];
estimatedPosefk             = [];
groundThruth                = [];


% Perform Simulation/Field experiments

%initial message
data_init = 1;

% Determine the number of messages in each topic
num_msgs = min([height(gps0_data.MessageList), height(gps1_data.MessageList), ...
height(gps2_data.MessageList), height(gps3_data.MessageList), height(rtk_data.MessageList)]);
num_msgs= min(num_msgs, 950);

% Pre-read all GPS and RTK messages dynamically
```

```
gps0_msgs = readMessages(gps0_data, data_init:num_msgs);
gps1_msgs = readMessages(gps1_data, data_init:num_msgs);
gps2_msgs = readMessages(gps2_data, data_init:num_msgs);
gps3_msgs = readMessages(gps3_data, data_init:num_msgs);
rtk_msgs  = readMessages(rtk_data, data_init:num_msgs);

% Preallocate arrays for efficiency
correctedAtt  = zeros(1, num_msgs-data_init+1);
velocities  = zeros(2, num_msgs-data_init+1);
error4gps = zeros(1, num_msgs-data_init+1);
errorEKF = zeros(1, num_msgs-data_init+1);
groundThruth = zeros(2, num_msgs-data_init+1);
dataGps0 = zeros(2, num_msgs-data_init+1);
dataGps1 = zeros(2, num_msgs-data_init+1);
dataGps2 = zeros(2, num_msgs-data_init+1);
dataGps3 = zeros(2, num_msgs-data_init+1);

for i = 1:num_msgs-data_init+1
    % Read Sensors and save ground truth

    msg0 = gps0_msgs{i};
    msg1 = gps1_msgs{i};
    msg2 = gps2_msgs{i};
    msg3 = gps3_msgs{i};
    msgRTK = rtk_msgs{i};

    % Save Ground truth
    [rtk_x, rtk_y]      = latlon2xy(msgRTK.Latitude, msgRTK.Longitude, origin(1), origin(2));
    rtk_xy              = [rtk_x; rtk_y] .* 1000;
    rtk_xy              = mtxRot * rtk_xy;
    rtk_x               = rtk_xy(1);
    rtk_y               = rtk_xy(2);
    groundThruth        = [groundThruth, [rtk_x; rtk_y]];

    % Process GPS data
    [gps0_x, gps0_y] = latlon2xy(msg0.Latitude, msg0.Longitude, origin(1), origin(2));
    gps0_xy = [gps0_x; gps0_y] .* 1000;
    gps0_xy = mtxRot * gps0_xy - [offset_x0; offset_y0];
    dataGps0(:, i) = gps0_xy;
    cov_gps0_x = msg0.PositionCovariance(1);
    cov_gps0_y = msg0.PositionCovariance(5);

    [gps1_x, gps1_y] = latlon2xy(msg1.Latitude, msg1.Longitude, origin(1), origin(2));
    gps1_xy = [gps1_x; gps1_y] .* 1000;
    gps1_xy = mtxRot * gps1_xy - [offset_x1; offset_y1];
    dataGps1(:, i) = gps1_xy;
    cov_gps1_x = msg1.PositionCovariance(1);
    cov_gps1_y = msg1.PositionCovariance(5);

    [gps2_x, gps2_y] = latlon2xy(msg2.Latitude, msg2.Longitude, origin(1), origin(2));
    gps2_xy = [gps2_x; gps2_y] .* 1000;
    gps2_xy = mtxRot * gps2_xy - [offset_x2; offset_y2];
    dataGps2(:, i) = gps2_xy;
    cov_gps2_x = msg2.PositionCovariance(1);
    cov_gps2_y = msg2.PositionCovariance(5);

    [gps3_x, gps3_y] = latlon2xy(msg3.Latitude, msg3.Longitude, origin(1), origin(2));
    gps3_xy = [gps3_x; gps3_y] .* 1000;
    gps3_xy = mtxRot * gps3_xy - [offset_x3; offset_y3];
    dataGps3(:, i) = gps3_xy;
    cov_gps3_x = msg3.PositionCovariance(1);
    cov_gps3_y = msg3.PositionCovariance(5);

    % Read attitude from Vectornav2
    imu2Data = readMessages(imu3, i);
    eulAngle = quat2eul([imu2Data{1,1}.Orientation.X, imu2Data{1,1}.Orientation.Y, ...
    imu2Data{1,1}.Orientation.Z, imu2Data{1,1}.Orientation.W]);
    th0 = -eulAngle(3);

    % Unwrap attitude
    delta = th0 - attitudeOld;
    if delta > pi
        correctFactor = correctFactor - 2 * pi;
    elseif delta < -pi
```

```
        correctFactor = correctFactor + 2 * pi;
    end
    attitudeOld = th0;
    attitude = th0 + correctFactor + attInit;
    correctedAtt(i) = attitude;

    % Read velocities
    vecOdomData = readMessages(vecOdom, i);
    linVelX = vecOdomData{1,1}.Twist.Twist.Linear.X;
    linVelY = vecOdomData{1,1}.Twist.Twist.Linear.Y;
    huskyLinVel = sqrt(linVelX^2 + linVelY^2) * sign(linVelX);
    huskyAngVel = vecOdomData{1,1}.Twist.Twist.Angular.Z;
    velocities(:, i) = [huskyAngVel; huskyLinVel];

    % Update measurements to mhe4gps and solve estimation
    pos0 = dataGps0(:, i);
    pos1 = dataGps1(:, i);
    pos2 = dataGps2(:, i);
    pos3 = dataGps3(:, i);


    updateMeasurement(mhe4gps, [correctedAtt(i); pos0; pos1; pos2; pos3]);
    updateInput(mhe4gps, velocities(:, i));
    solve(mhe4gps);
    q_k = mhe4gps.qk + alignFactor4gps;
    estimatedPos4gps = [estimatedPos4gps, q_k];

    %EKF
    predict(efkgps, FNt, jac_FNt, velocities(:, i));
    update(efkgps, [correctedAtt(i);  mean([pos0, pos1, pos2, pos3],2)], jac_hx);

    q_ekf = efkgps.Qtraj(:,end) + alignFactorefk;
    estimatedPosefk = [estimatedPosefk, q_ekf];

    error4gps(i) = sqrt((full(q_k(2)) - groundThruth(1, end))^2 + (full(q_k(3)) - groundThruth(2, end))^2);
    errorEKF(i) = sqrt((full(q_ekf(2)) - groundThruth(1, end))^2 + (full(q_ekf(3)) - groundThruth(2, end))^2);

end
dif_error = errorEKF(:)-error4gps(:);
disp('Error promedio MHE:')
disp(mean(error4gps(50:end)))
disp('Error promedio EKF:')
disp(mean(errorEKF(50:end)))
disp('Error inicial MHE:')
disp((error4gps(1)))
disp('Error inicial EKF:')
disp((errorEKF(1)))

beep;

first_i=20;

figure(1);
hold on;
grid on;
xlim([-18 22]);
ylim([-15 25]);
title('Datos GPS');
xlabel('X [m]');
ylabel('Y [m]');
daspect([1 1 1]);

plot(estimatedPos4gps(2, end), estimatedPos4gps(3, end), 'o', 'Color', 'r', 'MarkerSize', ...
7, 'MarkerFaceColor', 'r', 'DisplayName', 'Estimación MHE');
plot(groundThruth(1, end), groundThruth(2, end), 'o', 'Color', 'b', 'MarkerSize', ...
9, 'DisplayName', 'Ground Truth');
plot(full(estimatedPosefk(2, end)), full(estimatedPosefk(3, end)), 'o', 'Color', 'm', 'MarkerSize', ...
7, 'MarkerFaceColor', 'm', 'DisplayName', 'Estimación EKF');

plot(estimatedPos4gps(2, end-i+first_i:end), estimatedPos4gps(3, end-i+first_i:end), 'r', 'LineWidth', ...
1.5, 'DisplayName', 'Trayectoria MHE');
plot(groundThruth(1, end-i+first_i:end), groundThruth(2, end-i+first_i:end), 'b', 'LineWidth', ...
1.5, 'DisplayName', 'Trayectoria Ground Truth');
plot(full(estimatedPosefk(2, end-i+first_i:end)), full(estimatedPosefk(3, end-i+first_i:end)), 'm', 'LineWidth', ...
```

```
1.5, 'DisplayName', 'Trayectoria EKF');

legend('show', 'Location', 'best');
hold off;
drawnow;

figure(2);
hold on;
grid on;
title('Error de Posición');
xlabel('Iteración');
ylabel('Error [m]');
ylim([0 6]);

plot(error4gps(1+first_i-1:i), 'r', 'LineWidth', 1.5, 'DisplayName', 'Error MHE');
plot(errorEKF(1+first_i-1:i), 'm', 'LineWidth', 1.5, 'DisplayName', 'Error EKF');

legend('show', 'Location', 'best');
hold off;
drawnow;
```

Código A.2: Codigo para procesar datos recolectados

A.3 MHE

En esta sección se presenta el objeto de matlab usado como estimador para realizar los gráficos, el cual es complementario al código de Rosbag.

```
classdef mheMemNic < handle
    properties (GetAccess = public, SetAccess = private)
        % system's properties
        Ne, nq, nu, ny, Ts, Nt;
        % Numerical values of the optimal solution
        Qtraj, Utraj, Ytraj, Wtraj, Vtraj, q0bar, qkAux, qk, mtxW, mtxWPrm, ...
        mtxVPrm, mtxR, mtxP, v_kminusNe, sigmaP, cP;
        x0Val, y0Val, x1Val, y1Val, x2Val, y2Val, x3Val, y3Val;
        % Solver's parameters
        q_lb, q_ub, u_lb, u_ub;

        Alin, Blin, Clin, S, K, jac_fx, jac_fu, jac_hx, lastInput;

        Fsym, hsym;

        optSolver, optiMHE, solutionMHE;
        % MHE parameters:
        Y, U, qbar, P, Lh, alphaPrm, alphaPrmVal, cPrm, cPrmVal, fLossV1, jacV;
        JmheQ, JmheQVal;
        % MHE variables:
        Q, w, vAtt, vGps0, vGps1, vGps2, vGps3, X;
        vXYGps0, vXYGps1, vXYGps2, vXYGps3;
        x0, y0, x1, y1, x2, y2, x3, y3;
        % Parameters related to the GPS's positions
        wf, wr, l, h0, h1, h2, h3, th0, th1, th2, th3, normV1, normV2, thv;
        normV1Prm, normV2Prm, vNorm;
        th0Prm, th1Prm, th2Prm, th3Prm, h0Prm, h1Prm, h2Prm, h3Prm;
        % Quadratic (standard) MHE
        solutionMHEalpha, optiMHEalpha;
    end

    methods
        function obj = mheMemNic(Ne,Nt,F,h,dims,boxConst,Ts)
            obj.Ne              = Ne;
            obj.Nt              = Nt;
            obj.nq              = dims.nq;
            obj.nu              = dims.nu;
            obj.ny              = dims.ny;
            obj.Ts              = Ts;
```

```
            obj.Fsym            = F;
            obj.hsym            = h;
            obj.Qtraj           = zeros(obj.nq,obj.Ne+1);
            obj.Utraj           = zeros(obj.nu,obj.Ne);
            obj.Ytraj           = zeros(obj.ny,obj.Ne+1);
            obj.q0bar           = zeros(obj.nq,1);
            obj.qk              = zeros(obj.nq,1);
            obj.mtxP            = 1e6.*eye(obj.nq);
            obj.v_kminusNe      = ones(obj.ny,1);
            obj.lastInput       = zeros(obj.nu,1);

            obj.optSolver       = 'ipopt';

            obj.optiMHE         = casadi.Opti();
            obj.Y               = obj.optiMHE.parameter(obj.ny,obj.Ne+1);
            obj.U               = obj.optiMHE.parameter(obj.nu,obj.Ne);
            obj.qbar            = obj.optiMHE.parameter(obj.nq);
            obj.P               = obj.optiMHE.parameter(obj.nq,obj.nq);
            obj.Lh              = obj.optiMHE.parameter(obj.Nt,2);
            obj.mtxWPrm         = obj.optiMHE.parameter(obj.nq,obj.nq);
            obj.mtxVPrm         = obj.optiMHE.parameter(1+2*4,1+2*4);
            obj.normV1Prm       = obj.optiMHE.parameter;
            obj.normV2Prm       = obj.optiMHE.parameter;
            obj.th0Prm          = obj.optiMHE.parameter;
            obj.th1Prm          = obj.optiMHE.parameter;
            obj.th2Prm          = obj.optiMHE.parameter;
            obj.th3Prm          = obj.optiMHE.parameter;
            obj.h0Prm           = obj.optiMHE.parameter;
            obj.h1Prm           = obj.optiMHE.parameter;
            obj.h2Prm           = obj.optiMHE.parameter;
            obj.h3Prm           = obj.optiMHE.parameter;

            obj.optiMHE.set_value(obj.Y,obj.Ytraj);
            obj.optiMHE.set_value(obj.U,obj.Utraj);
            obj.optiMHE.set_value(obj.qbar,obj.q0bar);
            obj.optiMHE.set_value(obj.P,inv(obj.mtxP));

            obj.Q               = obj.optiMHE.variable(obj.nq,obj.Ne+1);
            obj.w               = obj.optiMHE.variable(obj.nq,obj.Ne);
            obj.vAtt            = obj.optiMHE.variable(1,obj.Ne+1);
            obj.vGps0           = obj.optiMHE.variable(2,obj.Ne+1);
            obj.vGps1           = obj.optiMHE.variable(2,obj.Ne+1);
            obj.vGps2           = obj.optiMHE.variable(2,obj.Ne+1);
            obj.vGps3           = obj.optiMHE.variable(2,obj.Ne+1);
            obj.vXYGps0         = obj.optiMHE.variable(2,obj.Ne+1);
            obj.vXYGps1         = obj.optiMHE.variable(2,obj.Ne+1);
            obj.vXYGps2         = obj.optiMHE.variable(2,obj.Ne+1);
            obj.vXYGps3         = obj.optiMHE.variable(2,obj.Ne+1);
            obj.vNorm           = obj.optiMHE.variable(2,obj.Ne+1);
            obj.X               = obj.optiMHE.variable(obj.nq);
            obj.x0              = obj.optiMHE.variable(1,obj.Ne+1);
            obj.y0              = obj.optiMHE.variable(1,obj.Ne+1);
            obj.x1              = obj.optiMHE.variable(1,obj.Ne+1);
            obj.y1              = obj.optiMHE.variable(1,obj.Ne+1);
            obj.x2              = obj.optiMHE.variable(1,obj.Ne+1);
            obj.y2              = obj.optiMHE.variable(1,obj.Ne+1);
            obj.x3              = obj.optiMHE.variable(1,obj.Ne+1);
            obj.y3              = obj.optiMHE.variable(1,obj.Ne+1);

            obj.JmheQ = (obj.X') * obj.P * obj.X;
            obj.optiMHE.subject_to(obj.X == obj.Q(:,1)-obj.qbar);
            for i=1:obj.Ne+1

                if i<obj.Ne+1
                    obj.JmheQ = obj.JmheQ + ( ((obj.w(:,i)') * (obj.mtxWPrm)) * (obj.w(:,i)) );
                    obj.optiMHE.subject_to( obj.Q(:,i+1) == obj.Fsym(obj.Q(:,i),obj.U(:,i)) + obj.w(:,i) );
                end

                obj.optiMHE.subject_to( obj.Y(1,i)   == obj.Q(1,i) + obj.vAtt(i) );
                obj.optiMHE.subject_to( obj.Y(2:3,i) == [obj.x0(i); obj.y0(i)] + obj.vGps0(:,i) );
                obj.optiMHE.subject_to( obj.Y(4:5,i) == [obj.x1(i); obj.y1(i)] + obj.vGps1(:,i) );
                obj.optiMHE.subject_to( obj.Y(6:7,i) == [obj.x2(i); obj.y2(i)] + obj.vGps2(:,i) );
                obj.optiMHE.subject_to( obj.Y(8:9,i) == [obj.x3(i); obj.y3(i)] + obj.vGps3(:,i) );
                % GPS's position constraints
```

```
            obj.optiMHE.subject_to( (obj.x0(i)-obj.x2(i))^2 + (obj.y0(i)-obj.y2(i))^2 == obj.normV1Prm^2 + obj.vNorm(1,i) );
            obj.optiMHE.subject_to( (obj.x1(i)-obj.x3(i))^2 + (obj.y1(i)-obj.y3(i))^2 == obj.normV2Prm^2 + obj.vNorm(2,i) );
            % GPS's measurements and center of mass constraints
            obj.optiMHE.subject_to( obj.x0(i) == obj.Q(2,i) + obj.h0Prm*cos(obj.Q(1,i) + obj.th0Prm) + obj.vXYGps0(1,i) );
            obj.optiMHE.subject_to( obj.y0(i) == obj.Q(3,i) + obj.h0Prm*sin(obj.Q(1,i) + obj.th0Prm) + obj.vXYGps0(2,i) );
            obj.optiMHE.subject_to( obj.x1(i) == obj.Q(2,i) + obj.h1Prm*cos(obj.Q(1,i) - obj.th1Prm) + obj.vXYGps1(1,i) );
            obj.optiMHE.subject_to( obj.y1(i) == obj.Q(3,i) + obj.h1Prm*sin(obj.Q(1,i) - obj.th1Prm) + obj.vXYGps1(2,i) );
            obj.optiMHE.subject_to( obj.x2(i) == obj.Q(2,i) + ...
            obj.h2Prm*cos(-pi/2 - obj.Q(1,i) + obj.th2Prm) + obj.vXYGps2(1,i) );
            obj.optiMHE.subject_to( obj.y2(i) == obj.Q(3,i) + ...
            obj.h2Prm*sin(-pi/2 - obj.Q(1,i) + obj.th2Prm) + obj.vXYGps2(2,i) );
            obj.optiMHE.subject_to( obj.x3(i) == obj.Q(2,i) + ...
            obj.h3Prm*cos(-pi/2 - obj.Q(1,i) - obj.th3Prm) + obj.vXYGps3(1,i) );
            obj.optiMHE.subject_to( obj.y3(i) == obj.Q(3,i) + ...
            obj.h3Prm*sin(-pi/2 - obj.Q(1,i) - obj.th3Prm) + obj.vXYGps3(2,i) );


            obj.JmheQ = obj.JmheQ + obj.vAtt(i) * obj.mtxVPrm(1) * obj.vAtt(i);
            obj.JmheQ = obj.JmheQ + obj.vGps0(:,i)' *obj.mtxVPrm(2:3,2:3) * obj.vGps0(:,i);
            obj.JmheQ = obj.JmheQ + obj.vGps1(:,i)' *obj.mtxVPrm(4:5,4:5) * obj.vGps1(:,i);
            obj.JmheQ = obj.JmheQ + obj.vGps2(:,i)' *obj.mtxVPrm(6:7,6:7) * obj.vGps2(:,i);
            obj.JmheQ = obj.JmheQ + obj.vGps3(:,i)' *obj.mtxVPrm(8:9,8:9) * obj.vGps3(:,i);

            obj.JmheQ = obj.JmheQ + 1e6 * (obj.vNorm(:,i)' * obj.vNorm(:,i));

            obj.JmheQ = obj.JmheQ + 1e6 * obj.vXYGps0(:,i)' * obj.vXYGps0(:,i);
            obj.JmheQ = obj.JmheQ + 1e6 * obj.vXYGps1(:,i)' * obj.vXYGps1(:,i);
            obj.JmheQ = obj.JmheQ + 1e6 * obj.vXYGps2(:,i)' * obj.vXYGps2(:,i);
            obj.JmheQ = obj.JmheQ + 1e6 * obj.vXYGps3(:,i)' * obj.vXYGps3(:,i);


            for j=1:obj.Nt
                obj.optiMHE.subject_to( obj.Q(j,i) == obj.Q(obj.Nt+j,i)-obj.Q(obj.Nt+j+1,i) );
            end

            xy_acum = [0;0];
            if obj.Nt>0
                for k=0:obj.Nt-1
                    xy_acum = xy_acum + [obj.Lh(end-k,1)*cos(obj.Q(obj.Nt+k+1,i)) ...
                    + obj.Lh(end-k,2)*cos(obj.Q(obj.Nt+k+2,i));
                    obj.Lh(end-k,1)*sin(obj.Q(obj.Nt+k+1,i)) + ...
                    obj.Lh(end-k,2)*sin(obj.Q(obj.Nt+k+2,i))];
                    aux_var = obj.Q(2*obj.Nt+2:2*obj.Nt+3,i) - xy_acum;
                    obj.optiMHE.subject_to( obj.Q(2*obj.Nt+1+(k+1)*2+1:2*obj.Nt+1+(k+2)*2,i) == aux_var );
                end
            end
        end
        obj.optiMHE.minimize(obj.JmheQ);
        p_optsMHE1 = struct('expand',true);
        s_optsMHE1 = struct('sb','no','print_level',0,'gamma_theta',1e-12,'jacobian_approximation','exact', ...
        'max_iter',10000,'tol',1e-8,'warm_start_init_point','no', ...
        'timing_statistics','no','print_timing_statistics','no','print_user_options', 'no', ...
        'print_options_documentation', 'no');
        obj.optiMHE.solver('ipopt',p_optsMHE1,s_optsMHE1);
    end

    function setPrmsGPSpos(obj,prmVals)
        obj.wf  = prmVals(1);
        obj.wr  = prmVals(2);
        obj.l   = prmVals(3);

        obj.h0      = sqrt((obj.wf/2)^2+(obj.l/2)^2);
        obj.h1      = obj.h0;
        obj.h2      = sqrt((obj.wr/2)^2+(obj.l/2)^2);
        obj.h3      = obj.h2;
        obj.th0     = atan(obj.wf/obj.l);
        obj.th1     = obj.th0;
        obj.th2     = atan(obj.wr/obj.l);
        obj.th3     = obj.th2;
        obj.normV1  = sqrt(((obj.wf+obj.wr)/2)^2+(obj.l/2)^2);
        obj.normV2  = obj.normV1;
        obj.thv     = 2*atan(obj.wf/obj.l);

        obj.optiMHE.set_value(obj.h0Prm,obj.h0);
```

```
            obj.optiMHE.set_value(obj.h1Prm,obj.h1);
            obj.optiMHE.set_value(obj.h2Prm,obj.h2);
            obj.optiMHE.set_value(obj.h3Prm,obj.h3);
            obj.optiMHE.set_value(obj.th0Prm,obj.th0);
            obj.optiMHE.set_value(obj.th1Prm,obj.th1);
            obj.optiMHE.set_value(obj.th2Prm,obj.th2);
            obj.optiMHE.set_value(obj.th3Prm,obj.th3);
            obj.optiMHE.set_value(obj.normV1Prm,obj.normV1);
            obj.optiMHE.set_value(obj.normV2Prm,obj.normV2);
        end

        function setVehicleDims(obj,dims)
            obj.optiMHE.set_value(obj.Lh,dims);
        end

        function setJacobians(obj,jacFx,jacFu,jacHx)
            obj.jac_fx = jacFx;
            obj.jac_fu = jacFu;
            obj.jac_hx = jacHx;
        end

        function setSigmaP(obj,sigma)
            obj.sigmaP = sigma;
        end

        function setCP(obj,c)
            obj.cP = c;
        end

        function updateACP(obj)

            obj.Alin = obj.jac_fx(obj.qk,obj.lastInput);
            obj.Blin = obj.jac_fu(obj.qk,obj.lastInput);
            obj.Clin = obj.jac_hx(obj.qk);

            m_aux    = expm([full(obj.Alin),full(obj.Blin);zeros(obj.nu,obj.nq+obj.nu)].*obj.Ts);
            obj.Alin = m_aux(1:obj.nq,1:obj.nq);
            obj.Blin = m_aux(1:obj.nq,obj.nq+1:end);

            obj.mtxP = obj.Alin*obj.mtxP*obj.Alin' + obj.mtxW;
            obj.S    = obj.Clin*obj.mtxP*obj.Clin' + obj.mtxR;
            obj.K    = obj.mtxP*obj.Clin'/(obj.S);
            obj.mtxP = (eye(obj.nq) - obj.K*obj.Clin)*obj.mtxP;
        end

        function updateMeasurement(obj,y)
            obj.Ytraj = [obj.Ytraj(:,2:end),y];
        end

        function updateInput(obj,u)
            obj.Utraj       = [obj.Utraj(:,2:end),u];
            obj.lastInput   = u;
        end

        function setMtxR(obj,mtxR)
            obj.mtxR = casadi.DM(mtxR);
            obj.optiMHE.set_value(obj.mtxVPrm,mtxR);
        end

        function setMtxW(obj,mtxW)
            obj.mtxW = casadi.DM(mtxW);
            obj.optiMHE.set_value(obj.mtxWPrm,mtxW);
        end

        function set_x0bar(obj,q0)
            obj.q0bar = q0;
            obj.optiMHE.set_value(obj.qbar,q0);
            obj.qkAux = q0;
        end

        function set_alpha(obj,alpha)
            obj.alphaPrmVal = alpha;
            obj.optiMHE.set_value(obj.alphaPrm,alpha);
        end
```

```
262
263         function set_cPrm(obj,c)
264             obj.cPrmVal = c;
265             obj.optiMHE.set_value(obj.cPrm,c);
266         end
267
268         function solve(obj)
269
270             obj.optiMHE.set_value(obj.Y,obj.Ytraj);
271             obj.optiMHE.set_value(obj.U,obj.Utraj);
272             obj.optiMHE.set_value(obj.qbar,obj.q0bar);
273
274             try
275                 obj.solutionMHE = obj.optiMHE.solve();
276
277                 obj.JmheQVal    = obj.solutionMHE.value(obj.JmheQ);
278
279                 obj.Qtraj       = obj.solutionMHE.value(obj.Q);
280                 obj.x0Val       = obj.solutionMHE.value(obj.x0);
281                 obj.y0Val       = obj.solutionMHE.value(obj.y0);
282                 obj.x1Val       = obj.solutionMHE.value(obj.x1);
283                 obj.y1Val       = obj.solutionMHE.value(obj.y1);
284                 obj.x2Val       = obj.solutionMHE.value(obj.x2);
285                 obj.y2Val       = obj.solutionMHE.value(obj.y2);
286                 obj.x3Val       = obj.solutionMHE.value(obj.x3);
287                 obj.y3Val       = obj.solutionMHE.value(obj.y3);
288                 obj.qk          = obj.Qtraj(:,end);
289                 obj.q0bar       = obj.Qtraj(:,2);
290                 obj.qkAux       = full(obj.Fsym(obj.qk,obj.lastInput));
291                 obj.v_kminusNe  = obj.Vtraj(:,1);
292                 updateACP(obj);
293                 obj.optiMHE.set_value(obj.P,inv(obj.mtxP));
294             catch
295                 obj.qk      = obj.qkAux;
296                 obj.qkAux   = full(obj.Fsym(obj.qk,obj.lastInput));
297             end
298         end
299
300     end
301
302 end
```

Código A.3: Implementación MHE en objeto de matlab