ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento de Ingeniería Mecánica, Térmica y de Fluidos

Área de Ingeniería Mecánica

TRABAJO FIN DE GRADO

PLATAFORMA INTERACTIVA PARA EL ANÁLISIS DE DATOS DE TELEMETRÍA EN MOTOCICLISMO UTILIZANDO MATLAB

Grado en

Ingeniería Mecánica

Autor: RUBÉN REINALDO PÉREZ

Tutor: MANUEL GONZALO ALCÁZAR VARGAS

Cotutor JAVIER PÉREZ FERNÁNDEZ

MÁLAGA, junio de 2.025

Tabla de contenido

Índice de figuras

Índice de tablas

1. Introducción

El proyecto nace de la necesidad de tener un programa de análisis de datos para el equipo UMA Racing Team, participante de la octava edición de MotoStudent representando la Universidad de Málaga.

1.1. MotoStudent

MotoStudent es una competición entre equipos de universidades de todo el mundo en la cual se fabrica una motocicleta que compite en el circuito de Motorland Aragón cada dos años. La finalidad es mostrar la destreza y conocimientos de los alumnos adquiridos en las escuelas durante los estudios universitarios.

Existen dos categorías que compiten por separado: eFuel y Electric. La primera utiliza motores de combustión alimentados por combustibles sintéticos mientras que la segunda utiliza un motor eléctrico alimentado por una batería.

La competición consiste de dos partes bien diferenciadas:

- Fase MS1: es la primera parte de la competición y está formado por un proyecto industrial que tiene que ser escrito por los miembros del equipo. En él, describimos quienes somos, cómo nos organizamos y trabajamos, las decisiones que hemos tomado en el desarrollo del prototipo junto a su debida justificación y un plan económico.
- Fase MS2: en esta fase se realizan numerosas pruebas al prototipo para evaluar ciertos aspectos técnicos y de seguridad. Están las pruebas estáticas en las que se verifican la seguridad estructural del prototipo, aislamiento eléctrico, etc. Y pruebas dinámicas como la gymkhana. La fase final de la competición consta de un fin de semana de carrera con entrenos, clasificación y carrera para determinar el ganador de esta fase.

El ganador será el que obtenga más puntos entre todas las pruebas del MS1 y MS2.

El UMA Racing Team se fundó en 2008, año en el que se disputó la primera edición de la competición y desde entonces ha competido en todas las ediciones menos en la de 2012. Hasta 2016 se competía con motos de combustión, ese mismo año se compitió con un prototipo eléctrico y otro de combustión, y desde entonces el equipo se ha centrado únicamente en los prototipos propulsados eléctricamente. El mejor resultado se alcanzó en 2018, victoria absoluta ganando el Best MotoStudent, premio

que se otorga al equipo que obtiene la mayor puntuación en su categoría. En 2021 se consiguió ganar la carrera.

Figura 1 – Carrera final de MotoStudent

La organización otorga a todos los equipos un Kit que incluye varios componentes que serán comunes entre todos los equipos de las respectivas categorías. En el caso de la categoría Electric, los componentes son:

- Motor eléctrico
- Juego de neumáticos delantero y trasero Bridgestone
- Bomba de freno y pinza de JJuan
- BENDER: dispositivo que garantiza el aislamiento eléctrico entre el chasis del prototipo y el sistema de alta tensión (High Voltage System)

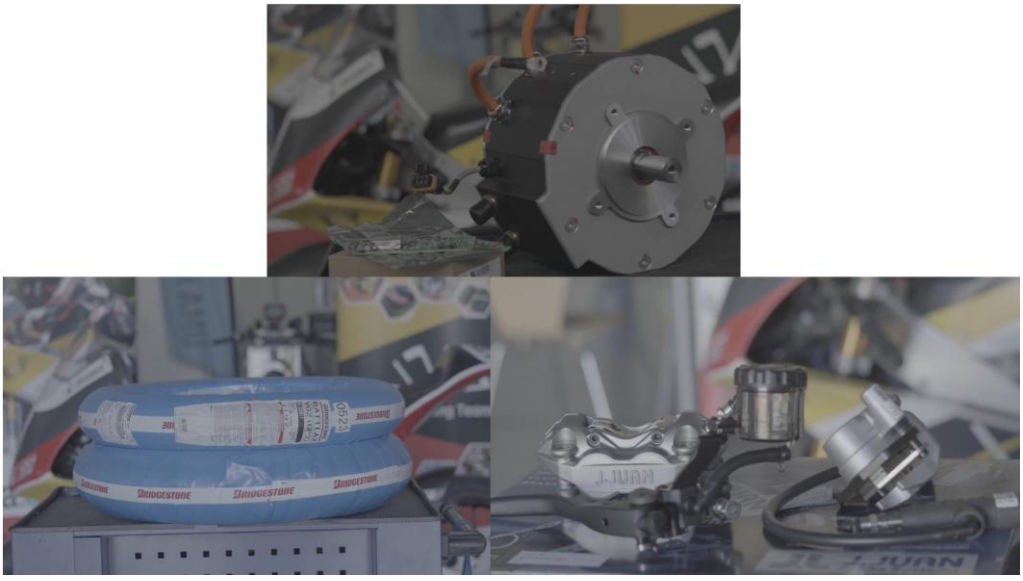

Figura 2 – Kit MotoStudent

1.2. Análisis de datos

En el MotorSport el análisis de datos es una parte fundamental dentro de un box.

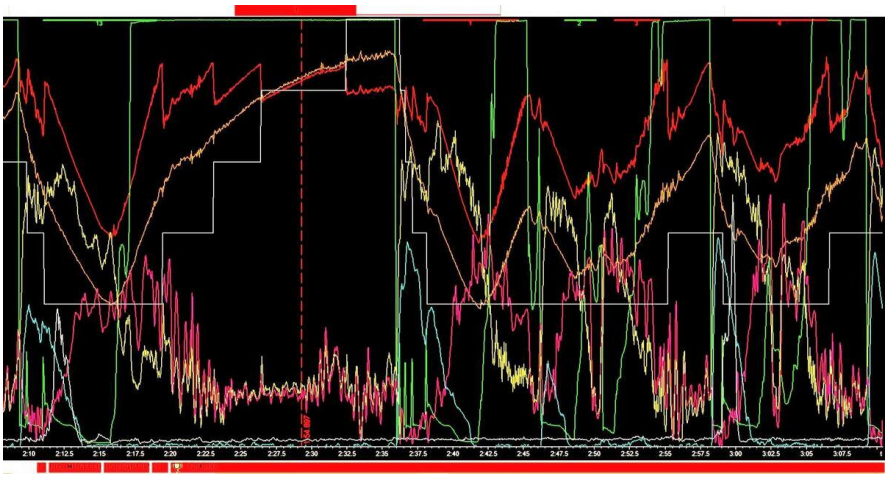

Figura 3 – Ejemplo de análisis de datos

En la F1 utilizan telemetría que consiste en adquisición de datos de forma telemática en directo, por tanto, pueden monitorizar las condiciones del monoplaza mientras está en uso. Esto no ocurre en MotoGP ni en MotoStudent, donde se realiza adquisición de datos, aunque coloquialmente también se le llame telemetría. En este documento, también aparece la palabra telemetría, aunque técnicamente sea análisis de datos. Estos datos se analizan una vez el prototipo ha entrado al box, lo que genera la casuística de tener que analizar de forma rápida los datos.

Estos datos son de gran interés. Bien analizados, nos pueden dar pistas sobre el estilo de pilotaje del piloto, el funcionamiento del prototipo, incluso nos puede dar la respuesta a la causa de una caída o un accidente. Igualmente, se pueden comparar datos de diferentes temporadas, diferentes años en el mismo circuito y comparar si hay mejora o no o comparar un mismo prototipo con piezas diferentes. Se justifica entonces la importancia de adquirir los datos de forma correcta y sobre todo de saber interpretarlos.

El problema es que son softwares fáciles de usar pero a veces difíciles de entender, ya que lo que ves son muchas gráficas que hay que saber interpretar de la forma correcta.

2. Objetivo

Uno de los objetivos del presente proyecto es que el programa sea fácilmente entendible y que lo pueda usar cualquiera mientras se lea el manual de usuario. Además, se va a intentar que sea lo más simple posible manteniendo los datos que consideramos de mayor importancia para la puesta a punto y comportamiento del prototipo. Por tanto, queda fuera del objetivo del presente proyecto análisis de temperaturas de funcionamiento o parámetros de funcionamiento del sistema eléctrico de la motocicleta. Queremos combinar lo mejor de los programas de análisis de datos y los de puesta a punto, como puede ser “Chassis Program”.

El objetivo del programa es combinar lo mejor de todos estos softwares y aplicarlo en un programa interactivo en el que, una vez procesados los datos, se puedan volcar las tandas en MATLAB y analizar. La idea es que el programa se use para analizar los datos entre tandas, para localizar posibles problemas y que ayude a tomar decisiones de puesta a punto o conducción del piloto.

3. Estado del arte

Los softwares de telemetría te muestran la información tal y como sale del sensor. En una gráfica se dispone el dato (o datos) a estudiar frente al tiempo. En el caso del motociclismo, la adquisición de datos se utiliza principalmente para conocer el comportamiento de las suspensiones, ángulo de inclinación, velocidad de giro del motor, velocidad del prototipo y presión de frenada, a grandes rasgos. En competiciones como MotoGP, tienen hasta 300 canales de información distintos ya que estos controlan más sistemas, que prototipos más simples como los de MotoStudent.

3.1. RaceStudio 2

Race Studio 2 es un software de telemetría clásico ampliamente utilizado en el ámbito de la competición, desarrollado por **AIM Sportline**. Su función principal es visualizar, analizar y gestionar los datos recogidos por el sistema de adquisición instalado en el prototipo, en este caso, una motocicleta de competición.

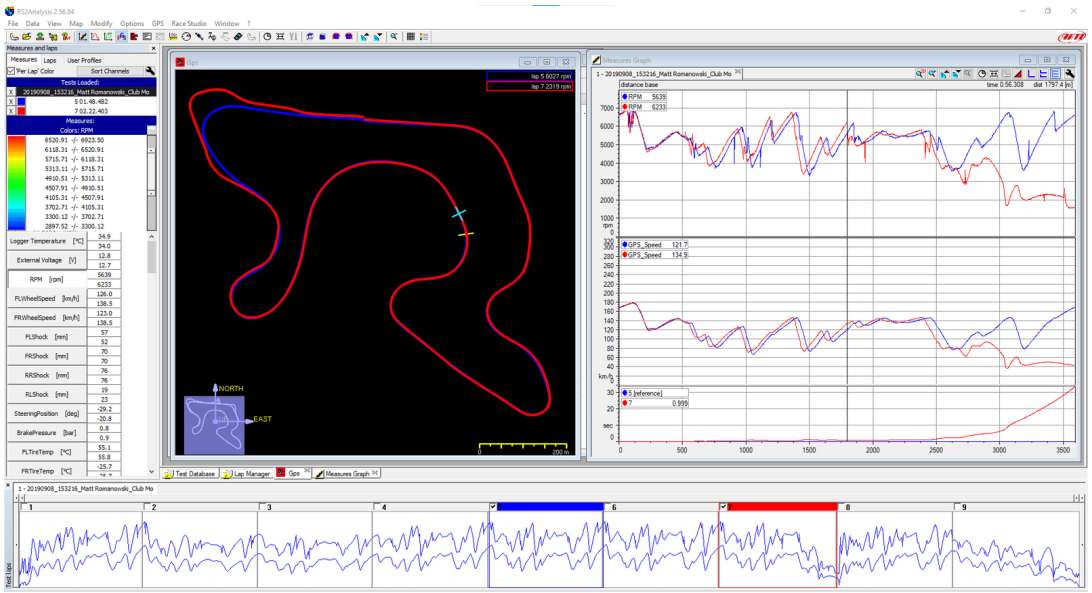

Figura 4 – RaceStudio 2

El software permite mostrar todos los canales de datos relevantes según las necesidades del equipo, incluyendo parámetros como el comportamiento dinámico del vehículo (velocidad, aceleración longitudinal y lateral, posición del acelerador, ángulo de dirección), recorridos de suspensión, régimen del motor, y temperaturas de distintos componentes (refrigerante, gases de escape, neumáticos, etc.). Además, posibilita el análisis de la técnica del piloto, identificando puntos de frenada, aperturas de gas, paso por curva y trazadas mediante datos GPS.

Race Studio 2 no solo actúa como visor, sino que también es una herramienta fundamental para la toma de decisiones en la puesta a punto de la motocicleta y la mejora del rendimiento en pista. Gracias a su interfaz intuitiva y la capacidad de superponer distintas vueltas o sesiones, facilita la comparación entre diferentes configuraciones, pilotos o condiciones de pista.

3.2. Chassis Program

Chassis Program es un software orientado específicamente a la **puesta a punto dinámica y geométrica de motocicletas de competición**. A diferencia de las herramientas de telemetría, que trabajan con datos adquiridos en pista, Chassis Program permite simular y analizar el comportamiento del chasis y las suspensiones en función de las configuraciones introducidas, siendo especialmente útil en la fase previa a la salida a pista o al momento de evaluar cambios de setup.

Este software trabaja a partir de una serie de datos geométricos del prototipo, como el ángulo de dirección, la distancia entre ejes, la longitud del basculante, las alturas delantera y trasera, las dimensiones del neumático, la relación de transmisión o la posición del centro de masas. A partir de esta información, calcula automáticamente parámetros clave como el **anti-squat**, **avance**, **ángulo de avance**, **altura del centro de gravedad**, entre otros. Además, permite ver cómo varían estos parámetros al introducir modificaciones en el setup, como un cambio de corona, ajuste en la precarga o modificación de la longitud del amortiguador trasero.

La interfaz ofrece una representación gráfica simplificada pero intuitiva del conjunto chasis-suspensión, junto con un panel lateral donde se introducen y visualizan los datos técnicos del setup. Una de las funciones más útiles del programa es la posibilidad de **guardar múltiples configuraciones** y compararlas entre sí, lo que facilita el análisis de los efectos de distintos cambios sobre el comportamiento dinámico de la moto.

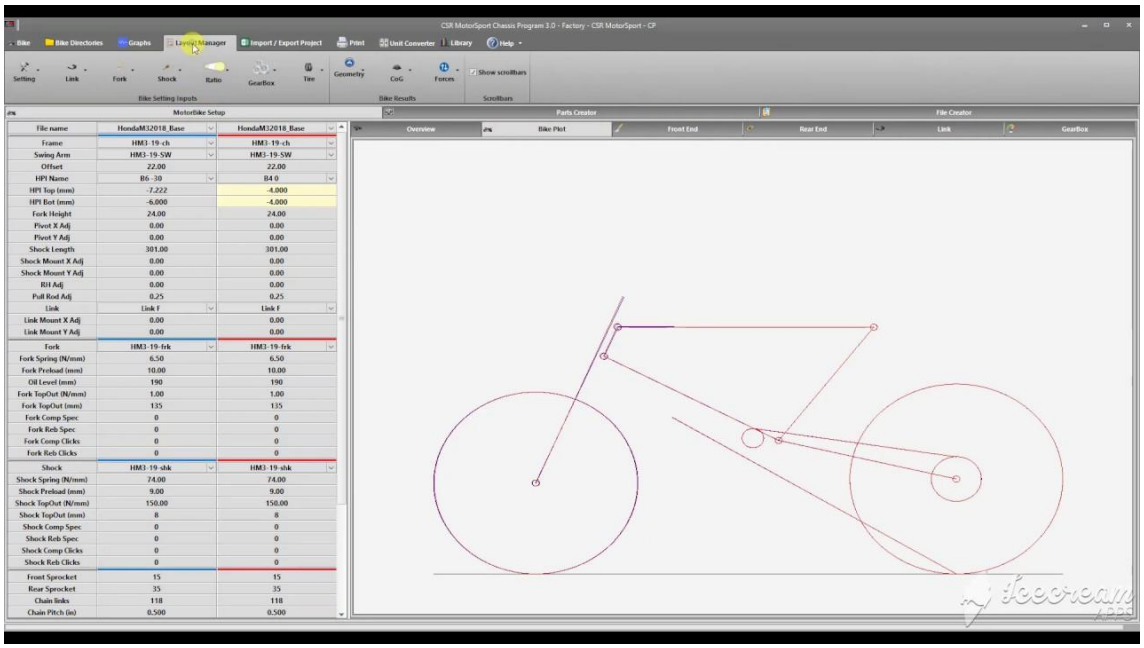

Figura 5 – Chassis Program

Adicionalmente, Chassis Program puede generar **gráficas de progresividad de suspensión**, **squat** y **leverage ratio**, herramientas clave para entender cómo trabaja la suspensión a lo largo de su recorrido y cómo se transmite la fuerza al neumático trasero durante las fases de aceleración, frenada o compresión. Esto permite una optimización más precisa de la configuración de suspensiones, algo fundamental en motocicletas de competición donde cada detalle puede marcar la diferencia en pista.

Como se puede observar, en el mundo de la telemetría y el análisis de rendimiento en motociclismo de competición, existen **dos categorías principales de software**, cada una con funciones bien diferenciadas pero complementarias entre sí.

Por un lado, están los programas de **análisis de datos adquiridos en pista**, como **Race Studio 2**, cuyo objetivo principal es interpretar la información registrada durante las sesiones reales. Estos softwares trabajan con datos crudos recogidos por sistemas de adquisición instalados en la motocicleta, tales como aceleraciones, velocidades, trayectorias GPS, uso del freno o gas, recorridos de suspensión, temperaturas, entre muchos otros. Su análisis permite estudiar en profundidad el comportamiento dinámico del prototipo y la técnica del piloto, identificar patrones, detectar problemas de rendimiento o estabilidad, y valorar la eficacia de las decisiones técnicas tomadas anteriormente.

Por otro lado, existen herramientas enfocadas a la **simulación y análisis estático del setup**, como **Chassis Program** o **MotoSpec**. Estos programas no dependen de datos recogidos en pista, sino que permiten al ingeniero introducir información geométrica y de configuración del vehículo para analizar su impacto sobre parámetros críticos como el centro de gravedad, el anti-squat, el avance (trail), la progresividad de las suspensiones o la relación de apalancamiento. De este modo, se convierten en herramientas de previsión y optimización, ya que ayudan a entender de antemano cómo va a comportarse la motocicleta frente a cambios de setup, permitiendo tomar decisiones más fundamentadas antes de rodar.

En conjunto, ambas herramientas cubren distintas fases del trabajo técnico: **Race Studio 2** es esencial en el análisis **post-sesión**, cuando ya se dispone de datos reales; mientras que **Chassis Program** resulta clave en la **fase de preparación**, ayudando a definir una base sólida sobre la que construir el rendimiento en pista.

4. Programas utilizados

4.1. MATLAB

MATLAB es un entorno de programación de alto nivel desarrollado por MathWorks, orientado al cálculo numérico, análisis de datos, simulación y visualización gráfica. Su estructura matricial y enfoque vectorizado lo hacen especialmente eficiente en aplicaciones científicas e ingenieriles.

El entorno cuenta con múltiples *toolboxes* especializadas, como control automático, procesamiento de señales, visión artificial e inteligencia artificial, además de integrar **Simulink**, una plataforma para la modelización y simulación de sistemas dinámicos mediante diagramas de bloques.

MATLAB es ampliamente utilizado en entornos académicos e industriales por su capacidad de prototipado rápido, análisis avanzado y conectividad con hardware y lenguajes externos como C/C++ o Python.

4.1.1. App designer

App Designer es una herramienta integrada en MATLAB que permite el desarrollo de interfaces gráficas de usuario (GUIs) de forma eficiente y estructurada. A diferencia de enfoques anteriores como GUIDE (ya descontinuado), App Designer combina un entorno de diseño visual con la posibilidad de escribir código MATLAB en un mismo espacio de trabajo, facilitando la creación de aplicaciones interactivas de forma profesional y con alto grado de personalización.

El entorno de App Designer está compuesto principalmente por dos áreas:

1. **La vista de diseño** (Design View), donde el usuario puede arrastrar y soltar componentes gráficos desde una biblioteca integrada. Estos componentes incluyen botones, sliders, menús desplegables, cajas de texto, interruptores, gráficas, tablas, paneles y contenedores, entre otros.

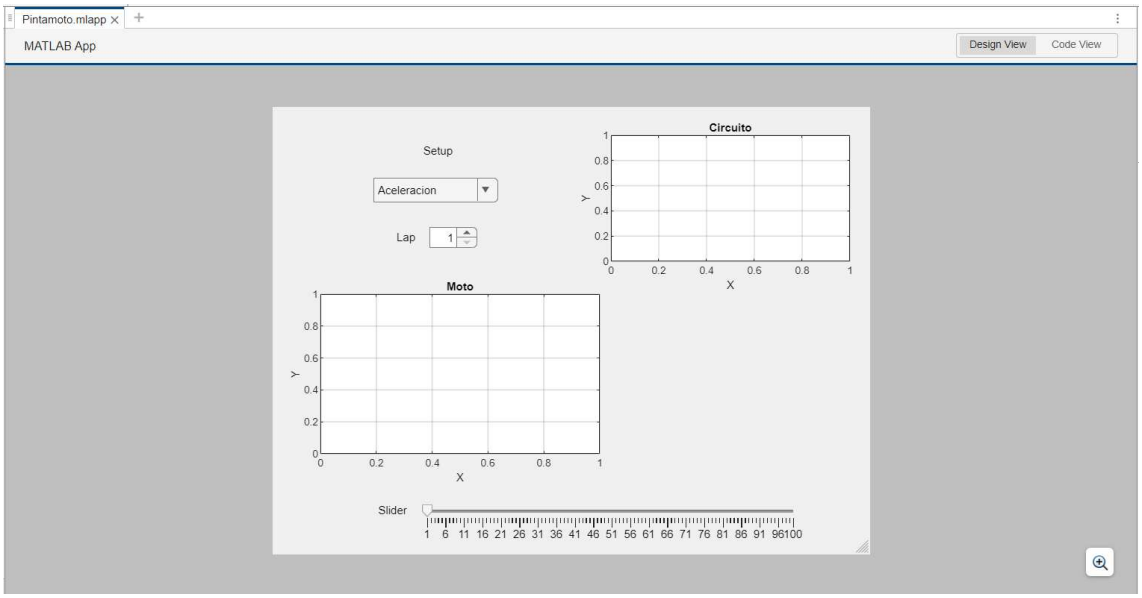

Figura 6 – Design view

2. **La vista de código** (Code View), donde se gestiona la lógica funcional de la aplicación mediante la escritura de scripts y funciones en lenguaje MATLAB. Aquí es donde se definen las **callbacks**, es decir, las funciones que se ejecutan cuando el usuario interactúa con un componente (por ejemplo, pulsar un botón o modificar un slider).

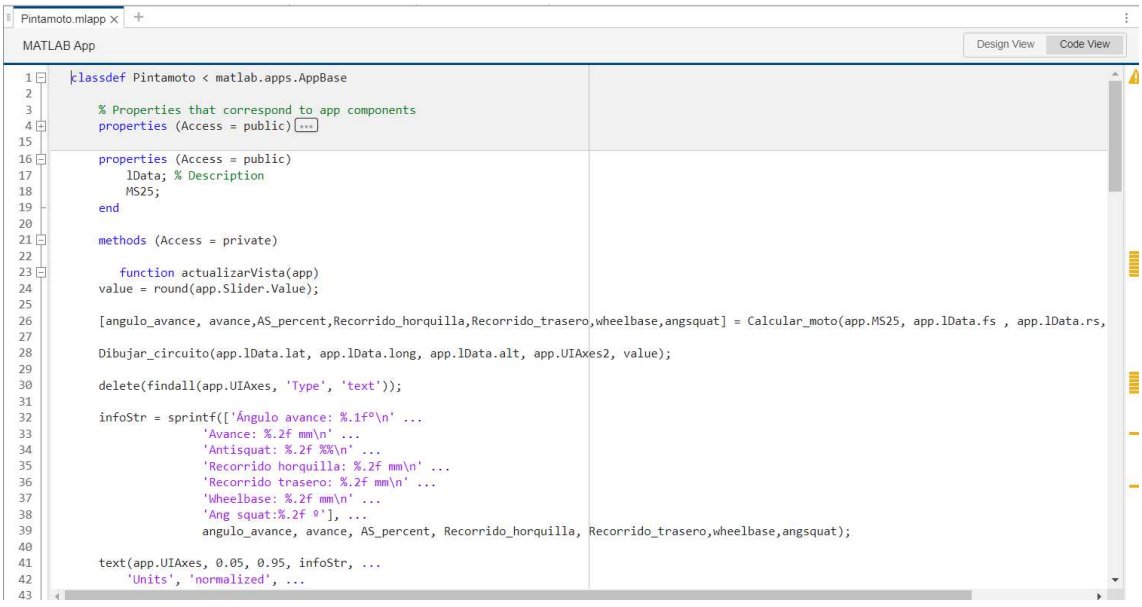

Figura 7 – Code view

Cada componente gráfico está asociado a propiedades y métodos que pueden modificarse dinámicamente durante la ejecución del programa. Además, App Designer organiza automáticamente el código en una **clase de tipo app**, lo que favorece la estructuración orientada a objetos y permite una mejor organización del

flujo de datos y funciones internas. Esta arquitectura permite una organización clara y modular del código, facilitando el mantenimiento y la escalabilidad del proyecto.

- **1. Properties (Propiedades)**

Las propiedades (properties) son variables definidas dentro de la clase app que pueden ser utilizadas y modificadas por cualquier método de la aplicación. Estas propiedades suelen usarse para almacenar:

- Datos persistentes (como estructuras cargadas por el usuario).
- Objetos de la interfaz gráfica (ejes, botones, sliders, etc.).
- Variables de estado (como el número de vuelta seleccionada o el set-up activo).

Definir correctamente las propiedades es clave para que distintos componentes y funciones de la interfaz puedan acceder y modificar información compartida de forma coherente.

- **2. Methods y callbacks**

Los methods son las funciones que definen el comportamiento de la app. Existen dos tipos principales:

- **Callbacks automáticos**: funciones que se ejecutan cuando el usuario interactúa con un componente (por ejemplo, al pulsar un botón, mover un slider, seleccionar un valor en un dropdown, etc.). App Designer crea automáticamente una plantilla de código para estas funciones cuando se añade un componente y se asocia un evento a él. Estas funciones tienen siempre como entrada la instancia de la app (app) y, en algunos casos, el evento (event), lo cual permite acceder a cualquier propiedad o método desde dentro del callback.
- **Funciones personalizadas**: definidas por el desarrollador para organizar tareas complejas o repetitivas (como calcular datos, limpiar gráficas, cargar archivos, etc.). Estas se suelen declarar como methods (Access = private) para mantener el encapsulamiento dentro de la clase.

- **3. startupFcn**

La función startupFcn es una callback especial que se ejecuta automáticamente **al iniciar la aplicación**. Su objetivo principal es inicializar la interfaz, cargar datos, establecer configuraciones por defecto y preparar el entorno gráfico. Se define dentro del bloque methods (Access = private) y se puede asignar como función de inicio desde el editor de diseño. Es un lugar ideal para cargar estructuras de datos

que usarán otras funciones, inicializar gráficas, definir límites de sliders, o configurar valores predeterminados.

App Designer es la plataforma elegida para desarrollar el programa.

4.2. SolidWorks

SolidWorks es un software de diseño asistido por ordenador (CAD) en tres dimensiones, desarrollado por Dassault Systèmes. Es ampliamente utilizado en ingeniería mecánica, diseño industrial, automoción, aeronáutica y otros sectores técnicos para la creación, simulación y documentación de modelos en 3D.

Este programa permite diseñar piezas, ensamblajes y dibujos técnicos con una alta precisión, facilitando el desarrollo de productos desde la fase conceptual hasta la fabricación. Entre sus funcionalidades más destacadas se incluyen el modelado paramétrico, análisis de elementos finitos (FEA), simulaciones de movimiento, estudios de esfuerzos y deformaciones, así como herramientas para la gestión del ciclo de vida del producto (PLM).

Gracias a su interfaz intuitiva y su integración con otras herramientas de diseño y simulación, SolidWorks se ha convertido en uno de los estándares de la industria para el diseño mecánico y la ingeniería de producto.

El software será usado para verificar que las funciones en MATLAB funcionan correctamente y se corresponden a lo que aparece en la aplicación, mediante un croquis de la geometría del prototipo (verificación de geometría).

4.3. Race Studio 2

Comentado en el apartado de Estado del arte, el software se utiliza para exportar datos de prototipos antiguos, que son los que se usan para el desarrollo y verificación de proyecto, ya que actualmente se encuentra en fabricación el úlitmo prototipo. Es el software usado por el UMA Racing Team para analizar los datos de telemetría.

5. Introducción a la adquisición de datos y geometría

En este apartado se va a introducir al análisis de datos y conocimientos básicos de geometría de una moto, con la intención de que cualquiera pueda entender el contenido técnico de este documento.

5.1. Conocimientos básicos de geometría

Lo primero de todo es saber de qué parte de la moto estamos hablando. Consideraremos la motocicleta desde el punto de vista de geometría en dos partes, parte delantera y parte trasera. La parte delantera es todo lo que se mueve con el manillar y la parte trasera el resto. Observar que cuando tengamos que hablar de las reacciones de la motocicleta en pista y tengamos que definir de que parte (delantera o trasera) estamos hablando tener en cuenta que la parte trasera empieza delante de nosotros (pipa de dirección o eje pivotante de dirección), esto nos ayudara a no confundir algunas reacciones que ocurren en la motocicleta que parecen que empiezan en la parte trasera y es en la parte delantera o viceversa.

Llamamos peso estático al peso total de la motocicleta en parado. También llamaremos reparto de pesos al porcentaje de apoyo de los ejes de rueda (o neumáticos), normalmente en modelos de motos de competición suele rondar un 40 % en el eje trasero y 60 % en el eje delantero. Interesa tener más peso en el tren trasero para generar más agarre de los neumáticos y tener más sensación del tren delantero al pilotar.

Llamamos peso dinámico al apoyo de los ejes de rueda (o neumáticos) delantero y trasero al frenar o acelerar producido por la transferencia de carga. En deceleración o frenada puede llegar hasta un 100 % de apoyo delantero (cuando se levanta la rueda trasera) y en una aceleración también podernos llegar a un 100 % de apoyo trasero (cuando se levanta la rueda delantera). El puño de gas en el paso por curva tiene especial importancia, pues debe de mantener el reparto de pesos dinámico (transferencia de carga) en el sitio que nos interesa y sin cambios bruscos. Una buena proporción en el paso por curva (antes de la aceleración de salida de curva) seria 40-45% eje delantero, 60-55 eje trasero para tener una buena direccionabilidad y un buen inicio de tracción. Hasta cierto límite podemos decir que cuanto más apoyo tenga el neumático más grip tendrá, luego podemos jugar con el reparto de peso para conseguir

más o menos tracción si aumentamos el grip en el neumático trasero, o más o menos capacidad de giro en el paso por curva si aumentamos el grip en el neumático delantero.

El reparto de peso en las motocicletas con piloto suele subir en el eje trasero y disminuir en el delantero entre un 15 a un 20 % en una moto de competición, por lo tanto, en competición podemos decir que el reparto de pesos estático con piloto es de un 40 a 41 % en el eje delantero y un 59 a 60 % en el eje trasero.

Como ya hemos dicho según los apoyos de los neumáticos en el asfalto, por deformación de carcasa estos tendrán más cantidad de goma en contacto y por lo tanto más grip (adherencia), esto es tanto como decir que, según el reparto de peso la moto tendrá más grip en la rueda delantera y menos en la trasera y viceversa.

Por otra parte, considerando que el reparto de peso estático es igual al dinámico sin aceleración ni deceleración (en velocidad constante, situación que se produce en algún porcentaje en la mayoría de las curvas al pasar de deceleración a aceleración) podemos decir, que el reparto de peso no solo influye en el apoyo de los neumáticos en el asfalto (grip), sino en el recorrido de suspensiones.

Si sabemos que el recorrido de suspensiones define la geometría de una motocicleta y que la geometría define el reparto de pesos, esto quiere decir que el reparto de pesos influye en la geometría y la geometría influye en el reparto de peso. Por todo lo anteriormente expuesto podemos afirmar que el reparto de pesos es importantísimo en el comportamiento de las motocicletas, porque de éste reparto de peso depende en gran medida la capacidad de frenada, capacidad de giro y capacidad de aceleración de nuestra motocicleta. [1]

- Peso suspendido y no suspendido (también llamado semisuspendido): Llamamos peso suspendido a todo lo que se mueve por encima de las suspensiones, es decir, con la moto parada y montándonos en ella, todo lo que reduce la distancia al suelo, es el peso suspendido. Llamamos peso no suspendido a todo lo que se mueve con las suspensiones, es decir, todo lo que soporta el movimiento de horquilla y basculante, como cubiertas, llantas, ejes de rueda, discos, pinzas, pastillas de freno, guardabarros, etc.

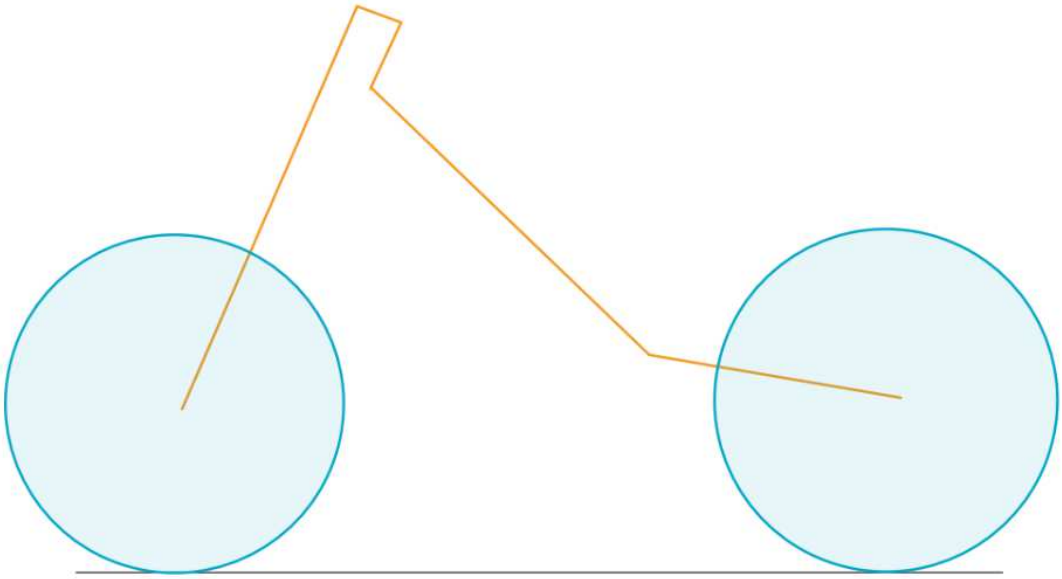

Figura 8 – Peso suspendido (amarillo) y no suspendido (azul)

El peso de las motocicletas de competición, como ya hemos indicado anteriormente, es muy importante. pero matizando en que parte de la motocicleta está dicho peso, tiene más o menos importancia. Si el peso está muy alto, nos beneficia para variar el centro de gravedad en las curvas y nos perjudica en todo lo demás, como en la transferencia de masas (peso dinámico), en las frenadas y aceleraciones.

El peso no suspendido es muy importante para el buen y estable comportamiento de las suspensiones (cuanto menos peso, mejor comportamiento de horquilla y amortiguador). Por ejemplo, el peso del conjunto llanta-neumático, es un peso importante por ser no suspendido, pero es mucho más importante por el efecto giroscópico del mismo.

- Avance: El avance, seguramente sea el concepto más influyente para la estabilidad direccional, junto con el efecto giroscópico de las ruedas. El avance se mide en milímetros y es la medida que existe entre el punto de contacto de neumático en el asfalto (o vertical del eje delantero al suelo) con el punto imaginario de contacto en el asfalto de la prolongación del eje de dirección (pipa de dirección).

Cuanta más distancia haya entre el punto de contacto del neumático en el asfalto con la intersección del eje de dirección al girarla, más fuerza autoaliniante, es decir, cuantos más milímetros de avance más fuerza autoaliniante tendremos. Resaltar que cuanto más avance más fuerza autoaliniante y por lo tanto el piloto notara una mayor

resistencia a girar la dirección, es decir, la dirección está más "dura·· y por este motivo tendremos la sensación de que la motocicleta pesa más, pero en realidad está más lenta de inclinar y de levantar.

- Distancia entre ejes: la distancia entre ejes es una de las medidas geométricas que influyen en gran medida en la estabilidad direccional de la motocicleta, por lo tanto, efectos como capacidad de giro, transferencia de carga (peso dinámico), estabilidad en rectas y curvas, apoyos de ejes delantero y trasero en frenadas y aceleraciones, centro de gravedad, etc., se ven afectado dependiendo si mayor o menor la distancia entre ejes. Como vemos esta medida geométrica es más importante de lo que en principio pensamos.

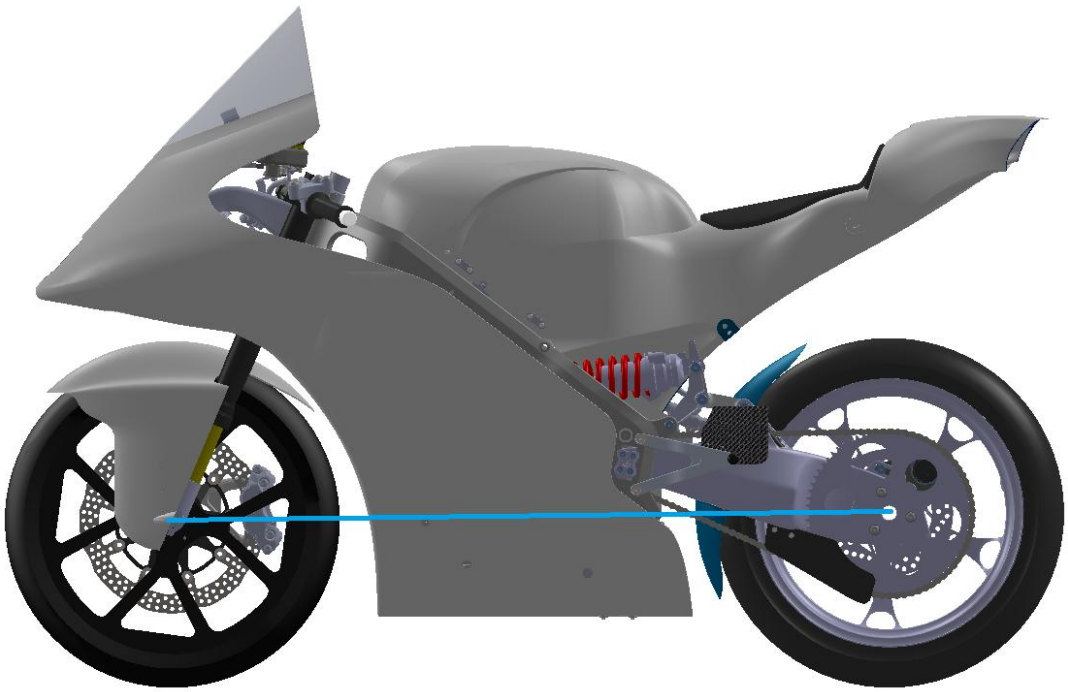

Figura 9 – Distancia entre ejes (wheelbase, línea azul)

Este concepto es bastante fácil de entender, si nosotros acortamos la distancia entre ejes, la motocicleta se hará un poco más ágil, tendremos una sensación de más ligera (menos peso), en curva lenta girara un poco mejor, el "derrapaje" trasero al acelerar será un poco más controlable porque digamos que tenemos una dirección más rápida; pero en contra la moto se hará más nerviosa de dirección en recta y curva rápida, la transferencia de masas (peso dinámico) lo notaremos más (subimos el centro de gravedad), en frenadas bruscas será un poco más difícil de controlar la rueda trasera, al acelerar perderemos el apoyo de la rueda delantera más fácilmente (importante), etc. Utilizando un símil de coche digamos que tiene un volante con menos desmultiplicado.

Si por el contrario alargamos la distancia entre ejes, la motocicleta se hará más estable de dirección, se comportara mejor en curva rápida y recta, la transferencia de masas (peso dinámico) se notara menos, en frenadas bruscas será un poco más estable, al acelerar mantendrá el apoyo del neumático delantero un poco más tiempo, en la salida tendré un poco menos tendencia a levantar rueda delantera; pero en contra la moto se hará un poco más pesada en curvas lentas, al acelerar el "derrapaje" trasero será un poco más difícil de controlar. También, aunque no es válido para competición, con una distancia entre ejes muy larga, como tenemos que dar más grados de giro a la dirección para modificar la trayectoria, la moto es mucho mas estable en línea recta a muy baja velocidad en complemento al bajo centro de gravedad efectivo que produce una distancia entre ejes muy grande (especialmente útil para motocicletas muy altas de peso).

- Centro de gravedad: Lo primero que tenemos que tener en cuenta es que el centro de gravedad (CG) es un punto en el espacio producido por la situación de la masa (peso moto-piloto). Para entender mejor la importancia del centro de gravedad, observaremos los tres movimientos posibles de una motocicleta en marcha: el cabeceo (penduleo), la inclinación y la guiñada (derrapaje).

En todos ellos tiene incidencia el centro de gravedad, de ahí la importancia de que el centro de gravedad (CG) esté alto, bajo, más adelante o más atrás en la motocicleta. Tendremos especial consideración que en los movimientos de cabeceo y guiñada los movimientos giran alrededor del centro de gravedad y en el de inclinación el punto pivotante el nivel del suelo.

Debemos tener siempre presente que cuando nos referimos al centro de gravedad (CG) podemos estar haciendo referencia al de la motocicleta sola o al que verdaderamente nos interesa. que es el CG conjunto de moto y piloto. En el futuro cuando nos refiramos al CG estamos considerando el conjunto de moto y piloto. El CG es el que nos va a influir en que la inercia de masas longitudinales (frenada y aceleración) sea mayor o menor, la inclinación sea lenta o rápida, mayor menor (para el mismo ángulo de giro), en que los movimientos de guiñada (derrapajes) sean más o menos bruscos y que necesitemos unos neumáticos más o menos anchos. [1]

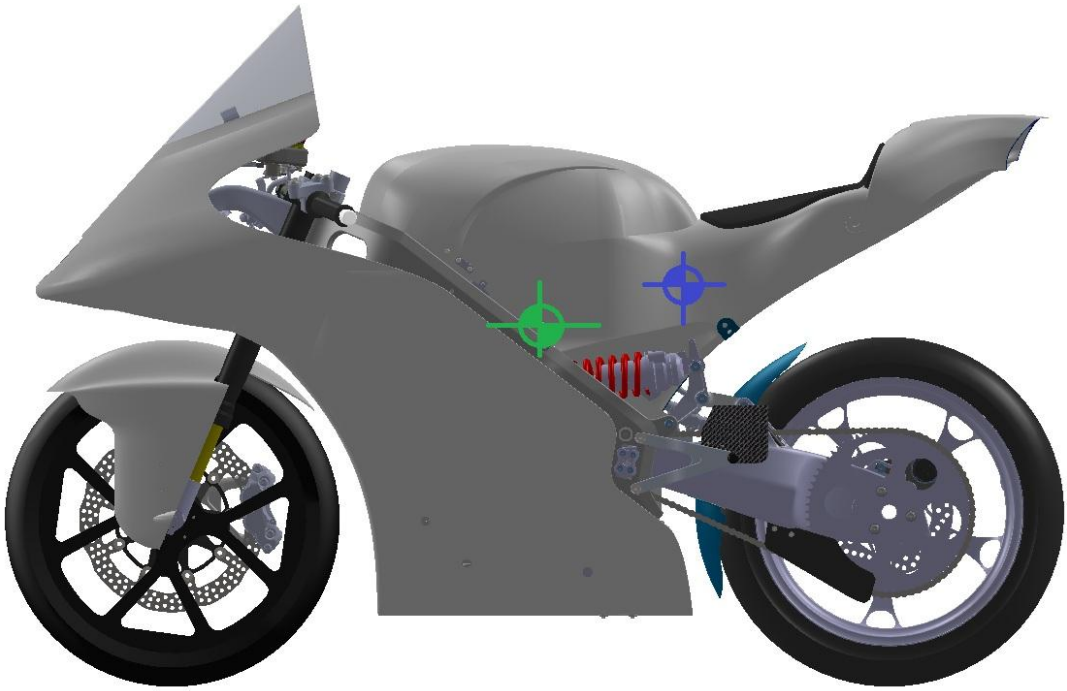

Figura 10 – CDG sin piloto (verde) y con piloto (morado)

El CG más adecuado de la motocicleta será diferente en cada situación y según otros factores como el ancho de los neumáticos. En general nos va a beneficiar que el CG esté lo más bajo posible en las frenadas y en las aceleraciones, que el CG sea alto nos va a beneficiar en el paso por curva para poder inclinar menos grados. El CG bajo hará que la motocicleta sea más rápida de inclinar (a la misma geometría), hará más suaves las inercias de masas longitudinales (acelerar y frenar), hará más fácil el mantenimiento del equilibrio a muy baja velocidad y por supuesto, tendremos mejor estabilidad en firmes irregulares.

Para contrarrestar la fuerza centrífuga necesitamos una inclinación del conjunto moto-piloto que produzca una fuerza vertical, como la fuerza vertical es mayor (a la misma inclinación) cuanto más alto este el CG, esto quiere decir que necesitaremos a la misma velocidad, peso y anchura de neumáticos, según el CG este más alto o bajo, necesitaremos inclinar menos y más respectivamente.

Para el nivel de inclinación no solo influye la altura del CG, también influye la anchura del neumático, ejemplo claro lo tenernos con la bicicleta y la motocicleta, a la misma velocidad la bicicleta con un CG mucho más alto y unos neumáticos mucho más estrecho necesitan mucha menos inclinación que una motocicleta. También considerar que al cambiar la geometría de la motocicleta no es lo mismo levantar de la parte delantera (horquilla y/o precargas) o bajar de la parte trasera (amortiguador o/y

precargas). Desde el punto de vista geométrico (avance, distancia entre ejes) y reparto de pesos es lo mismo, pero en un caso subimos el centro de gravedad y en otro lo bajamos.

- Transferencia de carga: Como ya hemos dicho anteriormente, llamamos peso cuando deberíamos llamar carga, ya que el peso es la atracción de la masa hacia el centro de la tierra producido por la fuerza de la gravedad, luego no podemos con aceleración o deceleración modificar el peso. Por lo tanto, cuando nos refiramos al peso, realmente lo que queremos decir es carga. Para ir simplificando conceptos diremos que la suma de las cargas (peso) verticales de los dos neumáticos es siempre la suma total del peso de la moto y el piloto. Cuando aceleramos se aumenta esta carga en el neumático trasero (eje trasero) en la misma proporción (o N) que disminuye en el delantero. y viceversa, cuando frenamos aumenta la carga en el neumático delantero los mismos N que disminuyen en el trasero.

En una transferencia de carga intervienen los siguientes elementos: la masa (peso de la motocicleta más el del piloto), la distancia entre ejes, el centro de gravedad (distancia al suelo en metros), la constante gravitatoria (g = 9,81 m/s^2) y el valor de la aceleración o deceleración.

Si queremos aminorar el efecto de la transferencia de carga, tanto en deceleración (frenadas) como en aceleración, debemos buscar lo siguiente: Bajar el centro de gravedad (CG) Aumentar la distancia entre ejes Reducir la masa (peso de moto-piloto)

La transferencia de carga tiene más importancia de la que en principio puede parecer. cuando hablamos de ella, siempre pensamos en las frenadas, pero influye también en las aceleraciones abriendo puño, pues según la posición del gas así tendremos los apoyos en el eje delantero y trasero, o lo que es lo mismo, así tendremos más o menos capacidad de tracción o más o menos capacidad de giro. Por otra parte, con la transferencia de carga podemos influir en la capacidad de la motocicleta de levantar rueda delantera en aceleraciones ó reducir la perdida de la rueda trasera en las frenadas, etc.

5.2. Adquisición de datos

En este apartado, se mostrará cómo se almacenan los datos y la interpretación de estos.

Hay una creencia general, sobre todo de los pilotos, que el SAD es "algo" que nos dice que debemos hacer en la motocicleta para que se comporte como necesitamos en pista. Nada más lejos de la realidad, el SAD, también conocido en DAQ (Data Aquisition System, solo nos dice que hace la motocicleta en pista según la utilización que hace el piloto al manejarla. Si la motocicleta hace o no hace lo que debe por mecánica, puesta a punto o/y por pilotaje es algo que se deduce del análisis e interpretación de los datos.

Vamos a definir de qué se compone un sistema de adquisición de datos:

- Data logger: unidad de almacenamiento que puede ser con o sin pantalla (en nuestro caso con pantalla).
- Display: es la pantalla por la que se muestra la información que se está guardando en la memoria.
- Cableado: es el que une los distintos sensores con el logger.
- Sensores: son los distintos aparatos que se encargan de medir y transmitir la información a la memoria del logger.

Figura 11 – Sensor de recorrido de horquilla delantera

- Beacon (emisor y receptor): es el sistema que tiene el SAD para saber cuando acaba y empieza una vuelta. Este sistema se usa cuando no tenemos GPS.
- Cable conector (Jack): cable que sirve para descargar la información de la memoria de almacenamiento al PC para procesarlo en el software. Hay algunos que se pueden descargar por Wi-fi.
- PC: ordenador con el software necesario para procesar y gestionar la información descargada de la memoria de almacenamiento.

Esto es lo necesario para comenzar con la adquisición de datos. Tenemos que entender, cómo se almacenan esos datos. [2]

Un SAD es una memoria donde se almacena la información que envían unos sensores. El sensor manda una información de un determinado valor para su almacenamiento en la memoria del *logger*. La frecuencia de muestreo que definamos para cada canal se determinara por la cantidad de memoria (Mb) que disponemos junto con la necesidad de información donde está instalado el sensor. Cuanto mayor sea la frecuencia de muestreo más fidelidad tendremos en la información de cada canal, pero menos tiempo de almacenamiento tendremos.

La memoria del *logger* es un almacenamiento en binario (base dos) y según sea su capacidad podremos almacenar más o menos datos, o lo que es lo mismo, podremos almacenar más o menos tiempo en pista, o lo que es lo mismo, en la misma cantidad de memoria podremos almacenar más o menos información según la frecuencia de muestreo.

La frecuencia de muestreo se mide en Hertzios (Hz), es decir, un Hz es una información por segundo. La memoria está compuesta de bytes (posiciones de memoria), 1 byte= 8 bit (binario, (base 2)).

Con las señales de los sensores (Hz) se pueden configurar canales matemáticos, los sensores ocupan un canal y pueden contribuir a construir otro. Cada canal tiene un consumo de Hz definido en la configuración con la frecuencia de muestreo y cada Hz tiene una ocupación en memoria de dos bytes, si tenemos un consumo de 500 Hz, tendremos una ocupación de memoria de 1000 bytes por segundo. Si queremos tener una capacidad de memoria capaz de almacenar 30 minutos (1800 segundos), ocuparemos 1.800.000 bytes (casi dos MB, 2.097.152 bytes). Ejemplo de unos consumos racionales de Hz para cada sensor podrían ser:

- Revoluciones por minuto (RPM): 10 a 50 Hz
- Velocidad: 20 a 50 Hz
- Acelerador: 10 a 20 Hz
- Temperatura agua: 1 Hz
- Suspensión: 200 Hz
- Cambio de marcha: 5 a 10 Hz

Como vemos, depende de la característica del dato a registrar. La suspensión necesita tanta frecuencia ya que está constantemente cambiando de posición y es de gran importancia para entender el comportamiento de la moto, en contraste, la

temperatura de agua se toma un dato por segundo porque, aunque es importante, la temperatura no cambia tan rápido como para necesitar mayor frecuencia de muestreo.

Según sea el tipo de *logger* tendrá más o menos canales analógicos (programables) y más o menos canales digitales. Cada sensor está en comunicación con un canal. Como hemos dicho anteriormente según el *logger* que tengamos dispondremos de más o menos canales analógicos y digitales. Los canales en la configuración del *logger* podremos tenerlos encendidos o apagados, según tengamos más o menos canales encendidos tendremos más o menos Hz de consumo. Esto es útil para cambiar la configuración según sea un entrenamiento para la puesta a punto, o una carrera de velocidad.

Una vez configurado el SAD y el software con los canales activos, los desactivados y los datos de interés es momento de analizar los datos.

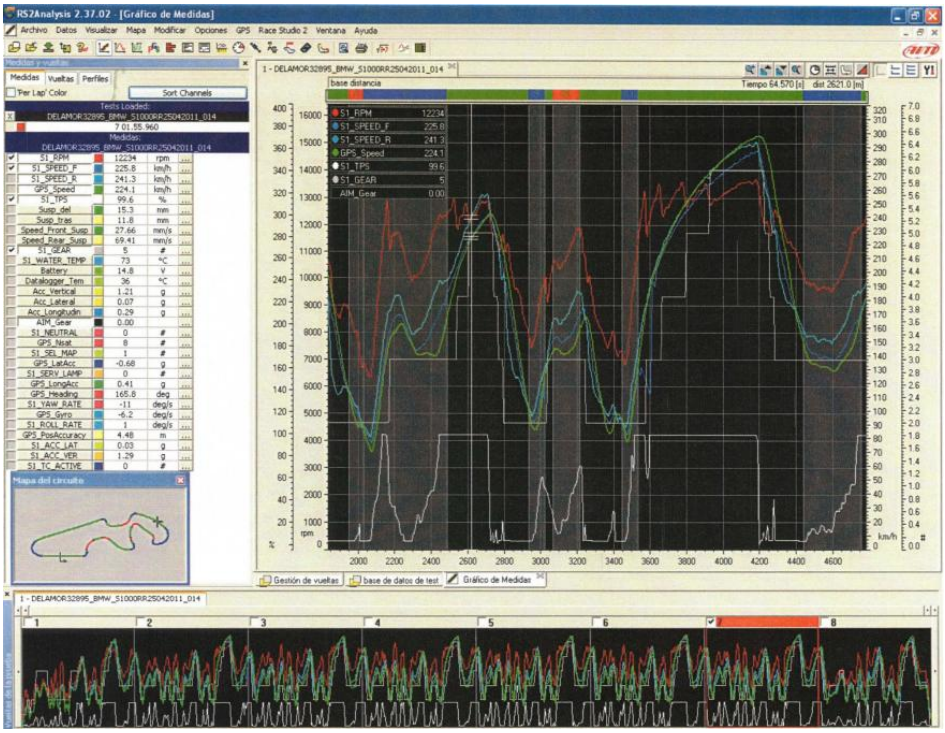

Figura 12 – Ejemplo de análisis de gráficos de velocidad, rpm, puño de gas y marchas

En los gráficos de un sistema de adquisición de datos (SAD), y con cierta experiencia podemos observar muchos datos del comportamiento de la moto. En este grafico donde tenemos solo unos datos básicos como son la velocidad, las revoluciones de motor, el acelerador y la marcha vamos a analizar su significado para irnos familiarizándonos con su interpretación.

Hay tres gráficos de velocidad. Hemos seleccionado las tres velocidades para poder analizar reacciones en aceleración, frenada y deformaciones a alta velocidad. En alta

velocidad los valores de los gráficos de velocidades de rueda delantera, trasera y GPS se distancian. Podemos observar que cada vez que el gráfico de TPS (acelerador) sube rápidamente de valor (se acelera) se produce una subida del grafico de RPM (el motor aumenta de revoluciones) una subida de los gráficos de velocidad delantera, trasera y GPS, pero también podemos observar que los tres suben de valor, pero con diferencias. Esto quiere decir que, a alta velocidad, aproximadamente a 220 km/h el neumático delantero aumenta de diámetros más que el trasero y por tanto da un valor un poco menor. También podemos observar que a partir de 290 Km/h hay un estancamiento de la velocidad indicado por el gráfico de la rueda trasera con respecto a los otros dos gráficos de velocidad (delantero y GPS), seguramente producido por aerodinámica (piloto erguido).

También podemos observar que a la altura del punto 2700 m hay una diferencia de valor a más alto el de velocidad de rueda trasera con respecto a los otros dos datos de velocidad (delantero y GPS), esto quiere decir que la rueda trasera está girando más deprisa que la moto (derrapada), esto lo deducimos porque está en curva y porque al gráfico de velocidad de rueda trasera le acompaña el gráfico de RPM.

En este ejemplo se puede ver un canal de las marchas de la moto, que no nos interesa estudiar porque el prototipo del UMA Racing Team tiene transmisión directa.

En el principio de las frenadas la rueda trasera disminuye la velocidad con respecto a la velocidad de la rueda delantera y la velocidad de GPS (bloqueo parcial de la rueda trasera). Vemos que se producen pequeños bloqueos de rueda trasera por falta de apoyo (en la frenada el peso dinámico de la moto pasa de atrás a delante (transferencia de carga)). Cuanta más diferencia de velocidad haya entre la rueda delantera y la rueda trasera en las frenadas más derivas y/o rebotes tendremos en el tren trasero.

En algunas aceleraciones la rueda delantera se levanta y pierde el contacto con el asfalto. Al acelerar el peso dinámico de la moto pasa de delante a atrás (transferencia de carga) y se produce un levantamiento y pérdida total del apoyo del neumático delantero con el asfalto. Esto produce en el gráfico de la velocidad de rueda delantera una estabilización de velocidad con una pequeña disminución rápida de velocidad. Si nos fijamos en los gráficos vemos que a la altura de los 3550 metros una interrupción brusca de la subida de valor del gráfico seguido de un aumento de velocidad rapidísimo, esto quiere decir donde empieza el despegue del neumático del asfalto y donde vuelve a tener contacto.

Por último, resaltar que, si analizando los datos rápidamente hemos podido deducir todo lo anteriormente expuesto, ya tenemos bastante utilidad en un SAD, pero si pensamos que todos estos datos los podemos comparar con otras vueltas u otros volcados anteriores o con otros pilotos, todos estos gráficos cobran mucho más valor para sacar conclusiones. Se demuestra el gran potencial de la adquisición de datos.

- Análisis de geometría dinámica con un ejemplo:

Para analizar los gráficos interpretando la *geometría dinámica* de la moto debemos tener en cuenta varias consideraciones. En primer lugar y más importante la *geometría estática*, es decir, la altura delantera de la moto que da la situación de las botellas (o barras en horquillas convencionales) sobre las tijas (o pletinas de dirección) y la altura trasera de la moto que da la longitud del amortiguador o la longitud de las bieletas de suspensión trasera. Esta *geometría estática* nos da el reparto de pesos de la moto, este dato es muy importante para interpretar la *geometría dinámica*.

Así como definir el recorrido y la velocidad de suspensiones es fácil estimar unos datos, podemos definir las suspensiones tienes recorridos muy cortos o muy largos, muy rápidos o muy lentos, los datos para interpretar la *geometría dinámica* están sujeta a muchos parámetros y distintas situaciones (frenada, paso por curva y aceleración). No podemos dar medidas fijas para considerar si la *geometría dinámica* es correcta o no, pero si podemos tener unos valores aproximados de relación entre el recorrido del eje delantero y el recorrido del eje trasero que nos ayuden, por lo menos, a saber, si la geometría es positiva (alta de delante o/y baja de detrás) o la geometría es negativa (baja de delante y/o alta de detrás).

También debemos considerar que los pilotos son muy personales a la hora de sentirse a gusto con la *geometría dinámica*, es decir, hay pilotos que les gusta una geometría más negativa para que la moto gire mejor pero tenga más problemas a la hora de acelerar y por el contrario otros pilotos le gustan una geometría más positiva donde tendrá más problema de giro pero más capacidad de aceleración.

En frenada, la horquilla delantera no debe hacer tope (físico o técnico, hace plano en el gráfico), es recomendable que siempre la horquilla tenga en su máximo recorrido la mayor absorción posible para no deformar en exceso la carcasa del neumático delantero. El recorrido aconsejable, como ya hemos dicho es alrededor del 90% del recorrido físico. El recorrido de suspensión trasera no debe quedar casi a 0 para no tener excesivos problemas de derivas o/y rebotes del tren trasero. Procurando conseguir

que estas dos condiciones en una frenada fuerte se cumpla podemos decir que estamos ante una *geometría dinámica* en frenada correcta.

En paso por curva, desde que soltamos frenos y justo antes de abrir gas, la situación de la horquilla delantera y el amortiguador trasero debe tener una diferencia de recorrido (de ejes) no más de 20 mm más baja la horquilla, es decir, si el amortiguador está a 25 mm de vástago (aproximadamente 55 mm de eje trasero dependiendo del amortiguador y el mecanismo) la horquilla debe estar por encima de 75 mm, y por supuesto, el vástago del amortiguador en el paso por curva debe estar por debajo de 30 mm. Tenemos que estar atentos a detectar "chatering" en el paso por curva, es bastante habitual y se puede producir por exceso y por defecto de apoyo en el asfalto del neumático delantero y en este momento (paso por curva) estamos intentando GIRAR en una curvo podemos cometer el error de atravesar la rueda delantera, es decir hacer unos grados de giros algo superior a los permitidos por la posición de la motocicleta y esto nos puede hacer que suframos el famoso "chatering".

En aceleración, la interpretación de los gráficos depende mucho del grip de las cubiertas y el grip de las cubiertas depende mucho de la situación geométrica de la moto (estática y dinámica) y por supuesto por el compuesto del neumático.

6. Descripción de las funciones de MATLAB

En este apartado se describen las distintas funciones de MATLAB que permiten funcionar a la aplicación. El flujo de información es el siguiente: la moto realiza tandas en circuito, estas se guardan en el dispositivo de adquisición de datos, se vuelcan estos datos en MATLAB y se convierten a `.mat`. Además, se definirá la geometría de la moto o diferentes geometrías a estudiar en diferentes archivos `.mat` y con esto el programa funcionaría.

Figura 13 – Flujo de trabajo

El programa tendrá funciones que se desarrollan en el entorno de MATLAB y se introducen en la app. Hay que minimizar todo lo posible la cantidad de código en la

propia app, por eso realizar todos los cálculos en funciones en las que entramos con variables definidas dentro de la app.

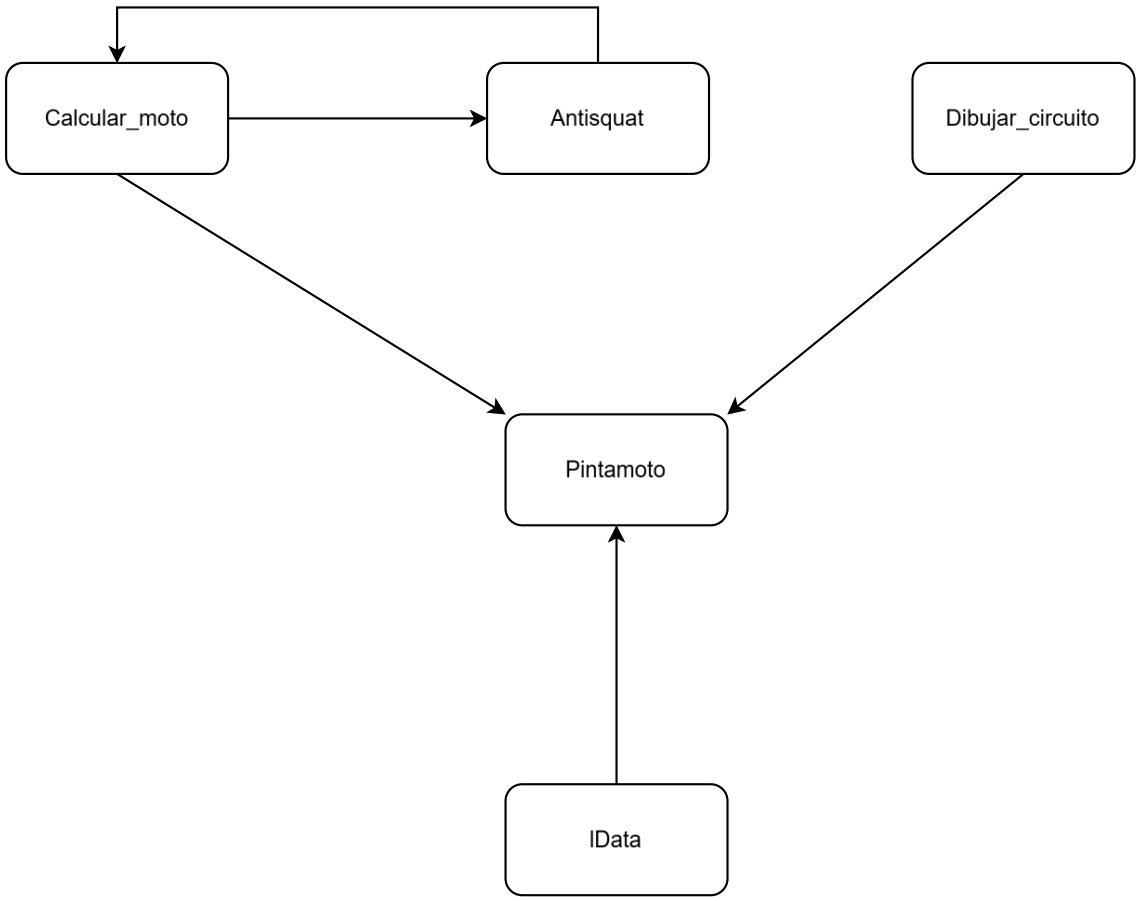

Figura 14 – Diagrama de flujo de funciones de MATLAB

Se necesita:

- Una función que calcule la geometría del prototipo en cualquier situación en función de los recorridos de suspensión. Estos son datos ya que vienen de los sensores ubicados en la suspensión de la moto. Se asume que la moto no hace caballitos ni invertidos (no despega las ruedas del suelo), esto simplifica el cálculo y rara vez en circuito la moto llega a esas situaciones por tanto es una simplificación aceptable.
- Otra función que grafique el circuito y un punto sobre él que represente en qué punto del circuito está la moto en el instante seleccionado.
- Calcular datos de geometría de interés para cada instante, como pueden ser el avance, ángulo de avance y distancia entre ejes.
- Función que calcule el porcentaje de antisquat.

Además, incluye un selector para cambiar de vuelta de tanda y otro para cambiar entre distintas geometrías definidas anteriormente en archivos `.mat`. Esto puede servir de comparación y de análisis de distintas geometrías.

6.1. convertirDatosTelemEstructura

Lo primero de todo es tratar los datos procedentes del AIM y hacerlos compatibles con MATLAB.

Para el programa en cuestión, los datos que interesan son: recorrido de suspensión delantera y trasera, latitud, longitud y altitud. El tiempo va implícito en los datos. Los recorridos de suspensión tienen una frecuencia de 20 Hz y los datos de posición, 10 Hz. Esto se ve reflejado en la longitud del vector de datos de una vuelta. Una vuelta más larga, contiene más puntos en el vector, ya que la frecuencia a la que toman los datos es constante.

En el espacio de trabajo de RaceStudio 2 hay una pestaña de exportar datos, le damos a exportar a MATLAB (explicado en el manual de usuario). Seleccionamos las vueltas que nos interesan y los datos mencionados anteriormente. Esto convierte los datos en una estructura de datos formada por vectores de una fila. Con esto ya podemos trabajar en MATLAB. Todas las funciones tendrán como entrada el dato de la posición en cualquiera de esos vectores de datos, lo que indica el tiempo.

Y este archivo `.mat` es el que procesa el script `convertirDatosTelemEstructura`.

Procesa el nombre de los datos extraídos de RaceStudio y los simplifica, todo bajo la estructura *lData* (el nombre se refiere a “lap data”). Y esta estructura la guarda con el nombre que le pongamos en el script.

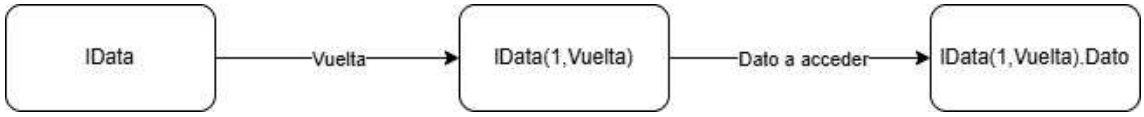

Figura 15 – Estructura de lData

De esta manera, resulta más sencillo acceder a los datos y quedan listos para procesarlos en la aplicación.

6.2. Calcular_moto

Esta es la función que calcula la geometría de la moto. Se considera la moto como eslabones de un mecanismo con ciertas restricciones y relaciones entre sí.

Partimos de los datos geométricos de la moto completamente extendida, estos son extraídos del software SolidWorks. Estos están recogidos en una estructura de datos, denominada bike en el caso de esta función. Los datos que aparecen a continuación son extraídos del prototipo de 2023 (actualmente el prototipo de 2025 está en fabricación), con una relación de transmisión de 13-70.

Tabla 1 – Información dentro de la estructura bike

Dato	Medida
front_wheel	288
fork_length	635
Yoke_offset	35
Steam_to_upyoke	140
Pivot_to_steam	655.6
Swingarm	466.79
Rear_wheel	300.65
Cdg_x	-583
Cdg_y	660
Pinion_x	-520
Pinion_y	410
pinon	13
corona	70

Hay campos que no se usan en la función, pero son extraídos igualmente de SolidWorks. Las incógnitas que tenemos son la distancia entre ejes, el ángulo de avance (o ángulo de cabeceo) y el ángulo entre el chasis y el basculante debido al movimiento del amortiguador trasero.

Para calcular el ángulo entre chasis y basculante, se introduce el mecanismo de suspensión trasera diseñada por un compañero del equipo en SolidWorks y en diferentes puntos del recorrido se toman pares de datos ángulo chasis basculante –

recorrido de amortiguador trasero. Para ello hay que sacar los datos de las suspensiones de la telemetría.

El recorrido de la horquilla es directamente el valor que aparece en la telemetría por tanto se puede mostrar el dato sin procesar, la salida es la entrada. En cambio, para el recorrido trasero se decidió mostrar el recorrido vertical de rueda trasera, que es un dato más clarificativo a la hora de entender cómo está funcionando el prototipo en vez del recorrido del amortiguador propiamente dicho. El AIM mide el recorrido del amortiguador, pero el sensor que lo mide no siempre tiene por qué estar calibrado y empezar en 0.

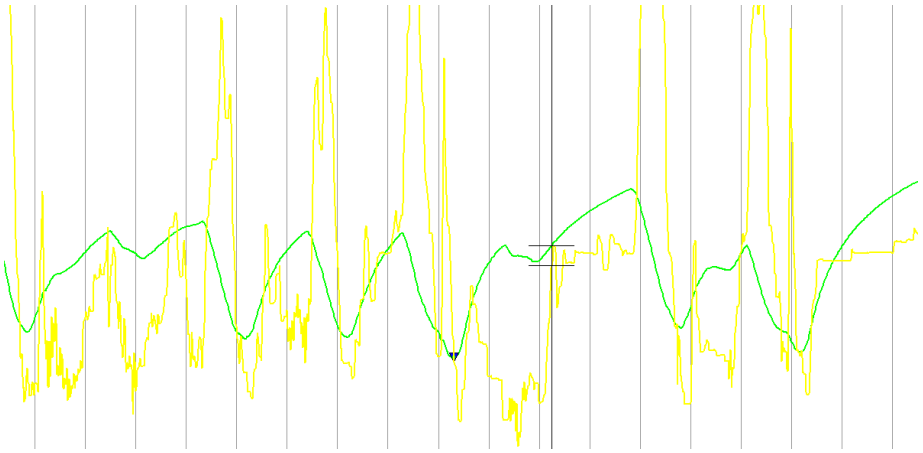

Figura 16 – Velocidad GPS (verde) y lectura de amortiguador trasero (amarillo)

Además, como se puede ver en la figura 15 cuando el prototipo frena (pendiente descendente de la línea verde) la lectura del amortiguador trasero aumenta, lo que quiere decir que el valor máximo que aparece en la gráfica corresponde al amortiguador completamente extendido, al contrario de lo considerado en la función `Calcular_moto`. Por ello, se le da la vuelta a la lectura y se considera el valor máximo leído en la telemetría como el amortiguador completamente extendido (razonable, ya que en frenada fuerte, el amortiguador se extiende casi por completo). Esto en la función queda de la siguiente forma:

$$Rt = Rear\_travel$$

$$Rt\,(i,1) = max\,(Rt) - Rt\,(i,1)$$

$$Rt\,(i,1) = 281 - Rt\,(i,1)$$

Siendo 281 la distancia entre ejes del amortiguador completamente extendido. Y con esto podemos calcular el ángulo chasis-basculante a través del recorrido del amortiguador trasero.

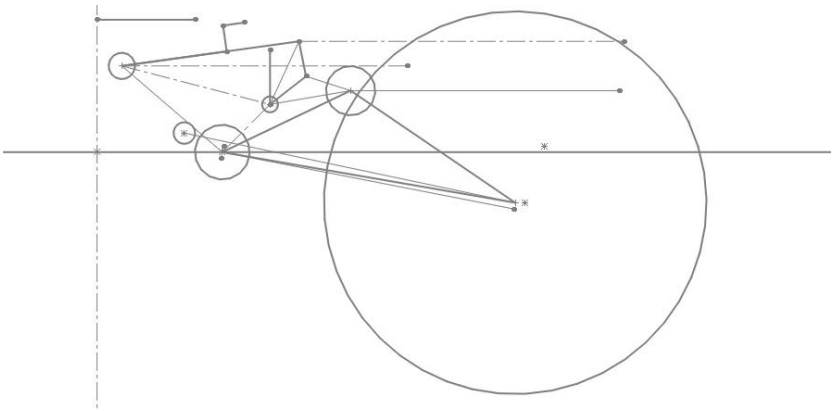

Figura 17 – Croquis de suspensión trasera en SolidWorks

Tomamos los pares de datos y los graficamos. Se obtiene una regresión lineal con MATLAB y se introduce en la función.

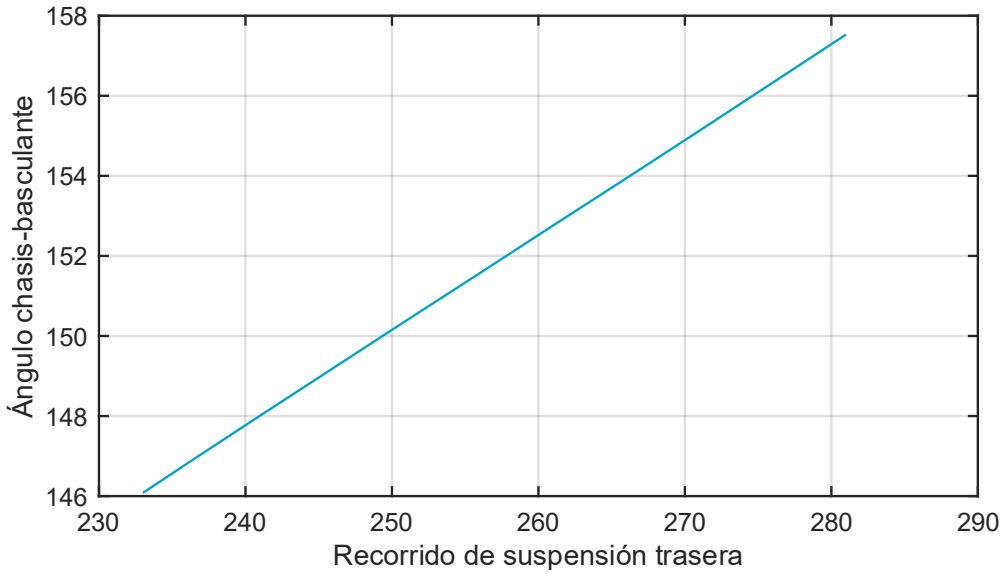

Figura 18 – Regresión lineal con los puntos obtenidos de SolidWorks

Y esta regresión es la que se introduce en la función para calcular el ángulo entre chasis y basculante en función del recorrido de la suspensión trasera que es conocida por el sistema de adquisición de datos.

De la misma forma, se calcula el recorrido vertical trasero para darlo como salida de la función. En SolidWorks se toman pares de datos de distancia entre ejes del amortiguador frente a recorrido vertical de rueda, se grafican los puntos y se realiza una regresión también lineal en este caso

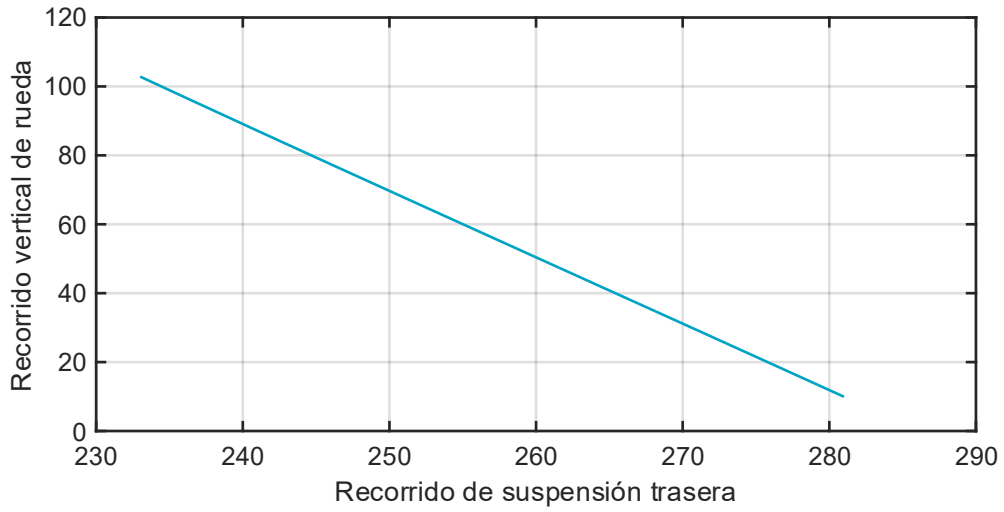

Figura 19 – Recorrido vertical de rueda frente a distancia entre ejes del amortiguador

Para calcular las coordenadas de los puntos considerados de la moto, usamos los datos de la estructura bike. Para simplificar los cálculos se considera que la horquilla de la moto se encuentra en posición vertical. Así, el ángulo de avance no es incógnita y es posible calcular las coordenadas de los puntos de la moto sin tener que resolver ningún sistema de ecuaciones. A continuación, se muestra la obtención de las coordenadas x, y:

$$fl = bike.fork\_length - Ft(i,1);$$

$$yk = bike.yoke\_offset;$$

$$su = bike.steam\_to\_upyoke;$$

$$ps = bike.pivot\_to\_steam;$$

$$sw = bike.swingarm;$$

$$x\_coor = [0,0, yk, yk, yk + ps * cosd(-9.26), yk + ps * cosd(-9.26) + sw * cosd(170.74)];$$

$$y\_coor = [0, fl, fl, fl - su, fl - su + ps * sind(-9.26), fl - su + ps * sind(-9.26) + sw * sind(170.74)];$$

Estas coordenadas representarían la moto como si estuviera haciendo un “invertido”, lo que no es correcto, la moto hay que representarla tocando con ambas ruedas el suelo (entiéndase que se represente la moto con ambas ruedas tocando el plano horizontal de la gráfica).

Para solucionar este problema, se calcula el ángulo de diferencia entre el eje de la rueda delantera y la rueda trasera. Se calcula mediante la coordenada vertical del punto del eje de la rueda trasera y la coordenada horizontal de ese mismo punto, ya que el origen de coordenadas se encuentra en el eje delantero.

$$\theta = \tan\left(\frac{y\_coor(6)}{x\_coor(6)}\right)$$

A este ángulo de rotación, que se restaría para que las ruedas hagan contacto con el plano horizontal, se le suma el ángulo que forma la diferencia de diámetro entre neumáticos, ya que el delantero tiene 576 mm de diámetro y el trasero es de 602 mm:

$$\varphi = \tan\left(\frac{bike.rear\_wheel\ -\ bike.front\_wheel}{wheelbase}\right)$$

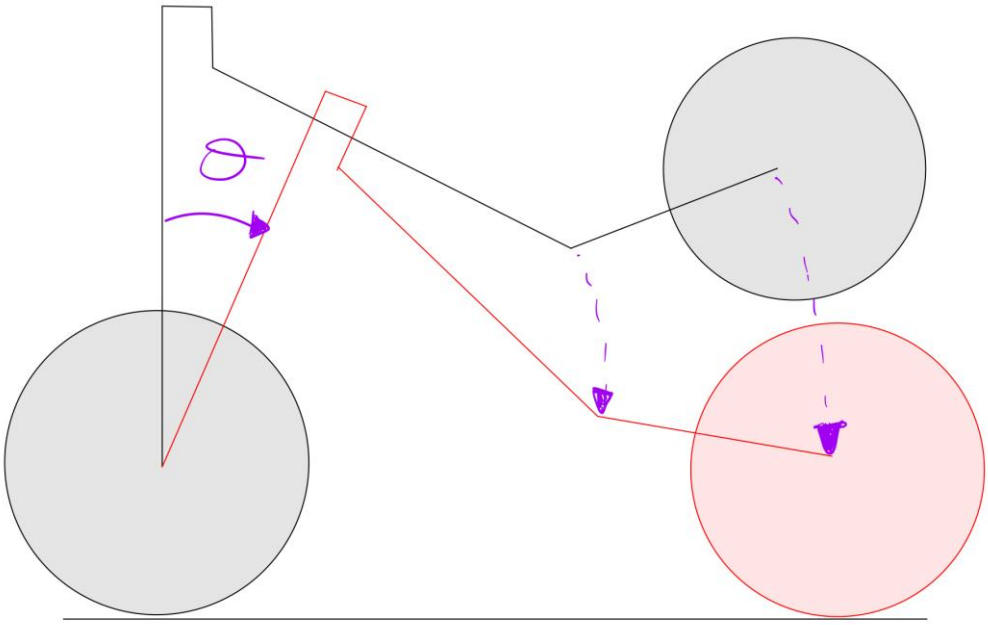

Figura 20 – Rotación de los eslabones que conforman la moto

Quedando así la matriz de rotación:

$$R = \begin{pmatrix} \cos -\theta + \varphi & -\sin -\theta + \varphi \\ \sin -\theta + \varphi & \cos -\theta + \varphi \end{pmatrix}$$

Si unimos los vectores de coordenadas en una matriz:

$$Coor\ = \begin{pmatrix} x\_coor \\ y\_coor \end{pmatrix}$$

Y multiplicamos por la matriz de rotación, ya quedan las coordenadas de la moto como queremos, listas para graficar:

$$Coor\_giradas = R * Coor$$

Con las coordenadas podemos calcular parámetros de geometría de interés, en este caso, consideramos de interés la distancia entre ejes, el ángulo de avance y el avance.

Para la distancia entre ejes es tan sencillo como dar el valor x de la rueda trasera, ya que el origen de coordenadas se encuentra en el eje delantero:

$$wheelbase = x_{rw} = Coor\_giradas\,(1,6)$$

El ángulo de avance se obtiene de la tangente de la distancia en x de eje de rueda delantera a la parte superior de la horquilla, entre la distancia en y de los mismos puntos:

$$\theta_{avance} = tan(\frac{x_{fork} - x_{fw}}{y_{fork} - y_{fw}}) = tan(\frac{Coor\_giradas(1,2) - Coor\_giradas(1,1)}{Coor\_giradas(2,2) - Coor\_giradas(2,1)})$$

Y para el avance es un poco más abstracto, pero no dejan de ser triángulos que con trigonometría se obtiene.

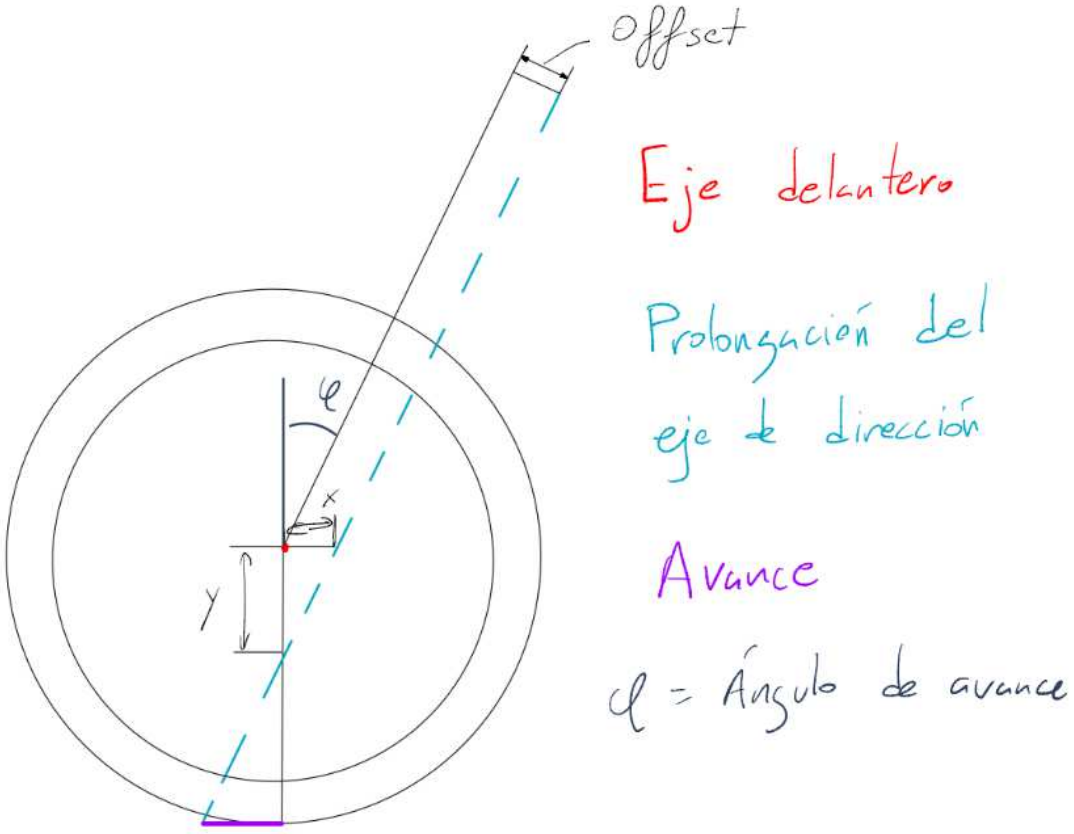

Figura 21 – Cálculo de avance

Nombrando “x” la distancia horizontal entre el eje delantero y la proyección del eje de dirección, y nombrando “y” la distancia vertical desde el eje delantero y la proyección del eje de dirección se obtiene lo siguiente:

$$x = \frac{offset}{cos(\theta_{avance})}$$

$$y = \frac{x}{tan(\theta_{avance})}$$

Siendo el avance la distancia perpendicular entre el eje de dirección y el eje que une las barras de horquilla. Con esto se obtiene el avance:

$$avance = tan(\theta_{avance}) * (y_{fw} - y)$$

En MATLAB:

$$avance = tand(angulo\_avance) * (bike.front\_wheel - y)$$

- Función Wheel: se incluye dentro de `Calcular_moto` debido a que simplemente es una función que dibuja un círculo. Es usada para dibujar las circunferencias de los neumáticos. Como entrada tiene coordenadas x e y del centro de la circunferencia y el radio, además de los *axes*, para que pueda aparecer graficada en la app.

6.3. Antisquat

El antisquat es un fenómeno generado debido al tirón de la cadena al acelerar, que hace que la suspensión trasera extienda o comprima. Este fenómeno depende de la posición del centro de gravedad, la posición del eje motor y el ángulo de la cadena, que depende de los dientes del piñón y la corona.

Este fenómeno se puede usar a nuestra ventaja. Las motos lo expresan mucho al ser un vehículo con un centro de gravedad alto y con una distancia entre ejes corta. En otro tipo de vehículos como los coches, con un centro de gravedad bajo y mayor distancia entre ejes, el antisquat no es de interés.

El antisquat afecta a la sensación del piloto y la forma en la que trabaja el neumático. Un antisquat por encima del 100 %, hace que el tren trasero en aceleración se extienda, lo que puede parecer contraintuitivo y para el piloto sería una sensación extraña, además de que estaríamos cargando de manera agresiva el neumático, lo que puede aumentar el desgaste del mismo. Como podemos ver, tiene una gran influencia

en el comportamiento de la motocicleta y, por ello, lo consideramos un parámetro de gran interés.

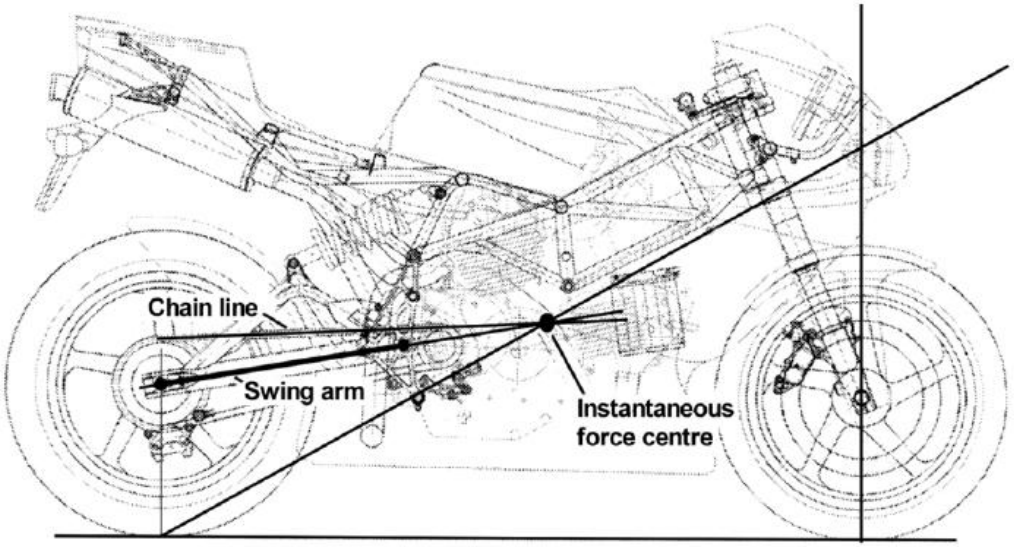

Figura 22 – Centro instantáneo de fuerzas [3]

Para el cálculo geométrico del antisquat necesitamos dibujar dos líneas desde el punto de contacto de la rueda trasera: la línea de squat, obtenida con el centro instantáneo de fuerzas y la línea de transferencia de carga.

El centro instantáneo de fuerzas se obtiene con dos líneas, una que se traza entre los ejes del basculante y la otra que es la prolongación de la rama superior de la cadena. La intersección de estas dos líneas nos da el centro instantáneo de fuerzas.

La línea de transferencia de carga es aquella que une el origen de la fuerza de tracción (punto de contacto de neumático trasero) con el centro de gravedad de la motocicleta. Determina cómo se realiza la transferencia de carga al acelerar o frenar. Esto afecta a la aceleración, la tendencia a hacer caballitos, estabilidad, etc.

Para calcular el antisquat hay dos maneras. En la primera, se traza una línea desde el punto de contacto del neumático trasero que pase por el centro instantáneo de rotación calculado y se extiende hasta que corte a una línea vertical trazada desde el eje delantero de la moto. La altura de este punto se compara con la altura del centro de gravedad de la motocicleta y esto nos da el porcentaje de antisquat.

La segunda manera (que es la usada en la función) es la división de la tangente del ángulo que forma la línea que une el punto de contacto con el centro instantáneo de fuerza, entre la tangente del ángulo de la línea que va del punto de contacto trasero, a

la línea que corta al eje delantero con la altura del centro de gravedad. Ambos métodos nos proporcionan el porcentaje de antisquat. [3]

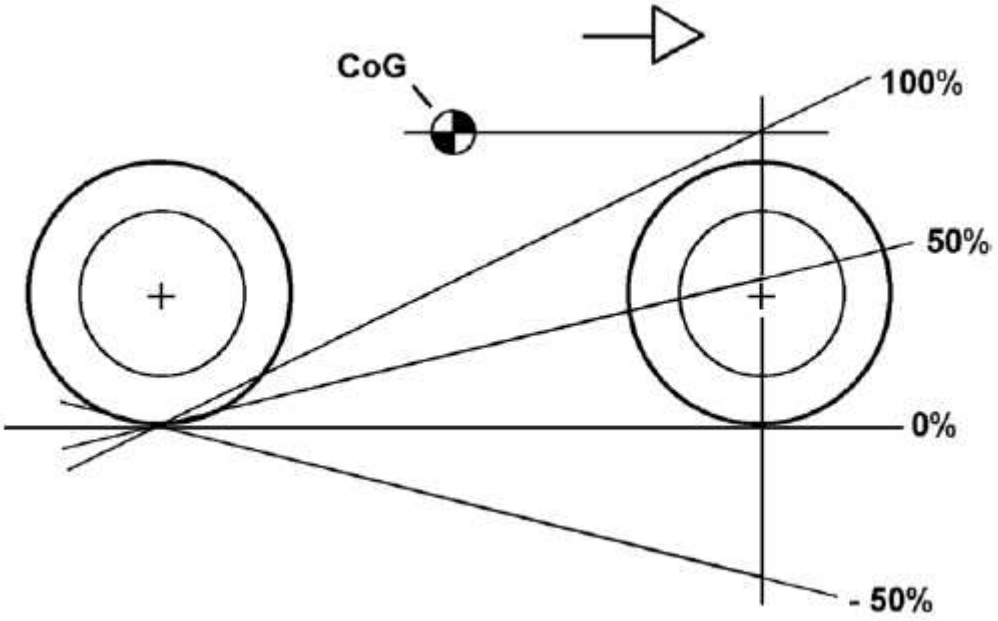

Figura 23 – Porcentaje de antisquat [3]

De la estructura bike se conoce la posición del centro de gravedad y del piñón del motor **en posición extendida**. No vale para la función, ya que, al comprimir y extender la moto en función del punto de la vuelta, el punto de centro de gravedad y del eje motor cambian constantemente.

Lo que se considera, es que el punto del centro de gravedad y del eje motor, **rotan y trasladan junto al chasis**, esto es lo que ocurre con el eje motor, pero con el centro de gravedad no exactamente. Es una aproximación válida, la coordenada x no se suele desplazar, la que sí se desplaza de forma considerable es la y. De esta manera, cuando la moto se comprime, el centro de gravedad baja, lo cual es correcto. Además, partimos de la base de que el dato del “cdg” es una aproximación, la moto no está fabricada a fecha de escritura del presente documento y el dato fue obtenido con las piezas que conocemos su masa y “cdg”, y con las más pesadas, despreciando las menos influenciables en el cálculo.

- Cálculo de las coordenadas del centro de gravedad y eje motor dentro de `Calcular_moto`:

Para obtener estos puntos y que se muevan solidarios al chasis, en SolidWorks se creó el croquis del prototipo en posición completamente extendida y se colocó el cdg.

Como punto de referencia se usó el centro superior de la pipa del chasis y se tomó dato de distancia entre puntos y ángulo respecto a la horizontal (coordenadas polares), tanto del centro de gravedad, como del eje motor.

$$x\_cdg\_pn = [463.28 * cos(5.49), \quad 622.17 * cos(337.47)]$$

$$y\_cdg\_pn = [fl + 463.28 * \sin(5.49), \quad 622.17 * sin(337.47)]$$

Y aplico la misma rotación que en `Calcular_moto`, ya que este punto se calcula de la misma forma:

$$cdg\_pn = [\, x\_cdg\_pn \,;\, y\_cdg\_pn \,]$$

$$cdg\_pn = R * cdg\_pn$$

Esta matriz contiene las coordenadas del centro de gravedad y el eje motor, y se mueven respecto al punto 4 de las coordenadas de la moto (respecto al chasis).

- Puntos de tangencia en el piñón y la corona:

  Con las coordenadas que nos interesan, hay que calcular los radios del piñón y la corona y los puntos de tangencia de la cadena con los mismos.

  Para el radio primitivo de las ruedas dentadas, tenemos como dato el paso de la cadena y el número de dientes (definido dentro de la estructura bike en bike.pinon y bike.corona)

$$r_{corona} = \frac{p_{cadena}}{2 * sin(\frac{180}{z_{corona}})}$$

  Y lo mismo para el piñón.

  Para comenzar el cálculo del antisquat, hacen falta las coordenadas del eje trasero y eje de basculación de la moto. Estos se obtienen de la función `Calcular_moto`, ya que, dependiendo del punto de la vuelta, estas coordenadas cambian y se calculan dentro de la función mencionada.

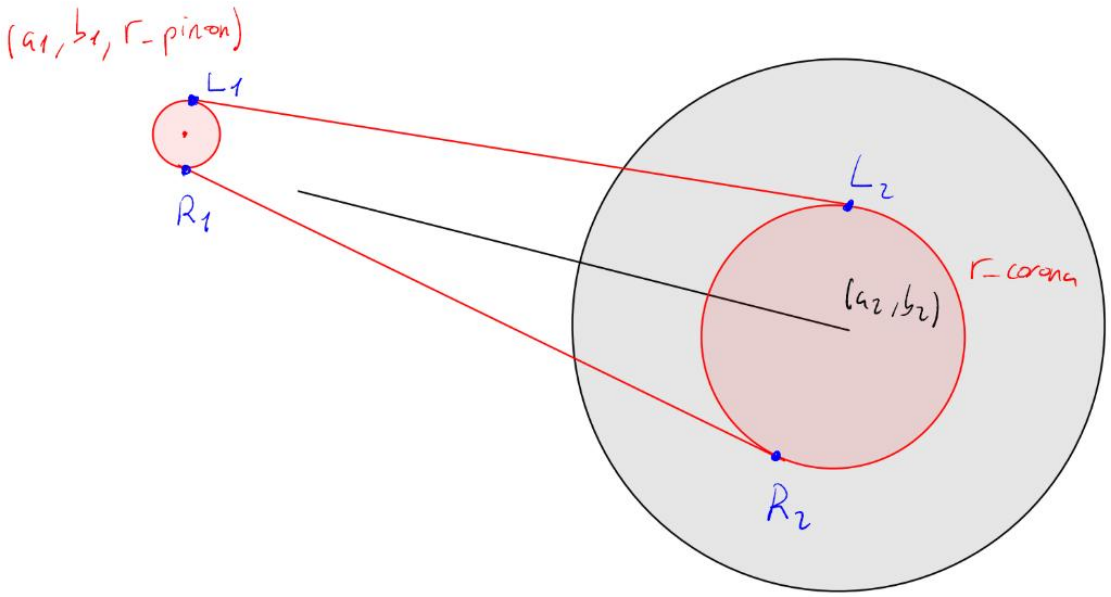

Figura 24 – Representación gráfica de los puntos a calcular

Se calcula la distancia entre piñón y corona:

$$D = (a_2 - a_1)^2 + (b_2 - b_1)^2$$

Donde a1, b1, a2 y b2 son las coordenadas del piñón y la corona como se puede ver en la figura.

Se obtienen proyecciones para calcular los vectores unitarios de las cadenas:

$$R = \frac{r_{corona} - r_{piñón}}{D}$$

$$S = \sqrt{\frac{D - (r_{corona} - r_{piñón})^2}{D}}$$

Y ya podemos calcular los vectores unitarios de las ramas de cadena y los puntos de tangencia deseados:

$$u\_sup = [-dx .* R - dy .* S; -dy .* R + dx .* S]$$

$$u\_inf = [-dx .* R + dy .* S; -dy .* R - dx .* S];$$

$$L1 = [a1', b1'] + r\_pinon * u\_sup'$$

$$L2 = [a2', b2'] + r\_corona * u\_sup'$$

$$R1 = [a1', b1'] + r\_pinon * u\_inf'$$

$$R2 = [a2', b2'] + r\_corona * u\_inf'$$

Con estos puntos se definen las ecuaciones de las rectas de las ramas de la cadena. Para el cálculo de porcentaje de antisquat únicamente nos interesa la rama superior, pero también se calcula la de la rama inferior, para poder graficarla.

Para la recta de la cadena, la pendiente:

$$m_{cadena} = \frac{L1_y - L2_y}{L1_x - L2_x}$$

Y la ordenada en el origen:

$$n_{cadena} = L2_y - m_{cadena} * L2_x$$

Respecto a la recta del basculante, se calcula mediante el ángulo del basculante, siendo (x_{pv}, y_{pv}) el punto de pivote del basculante y (x_{rw}, y_{rw}) el punto del centro de la rueda trasera:

$$\theta_{sw} = arctan(\frac{y_{pv} - y_{rw}}{x_{pv} - x_{rw}})$$

La pendiente:

$$m_{sw} = tan(\theta_{sw})$$

La ordenada en el origen:

$$n_{sw} = y_{rw}$$

Quedando definidas las rectas, hay que calcular el punto de intersección de ambas, en el código:

$$x_{squat} = \frac{n_{cadena} - n_{sw}}{m_{cadena} - m_{sw}}$$

$$y_{squat} = m_{sw} * x_{squat} + n_{sw}$$

Obtenemos el ángulo de squat:

$$\theta_{squat} = tan(\frac{y_{squat}}{x_{squat}})$$

Y si obtenemos el ángulo de transferencia de carga:

$$\varphi_{transfer} = tand(\frac{y_{cdg}}{wheelbase})$$

Finalmente obtenemos el porcentaje de antisquat:

$$AS_{\%} = tan(\frac{\theta_{squat}}{\varphi_{transfer}})$$

Estas ecuaciones se desarrollan en la función, que se encuentra en el anexo de códigos. Con SolidWorks se comparó en varias posiciones el valor obtenido por la función, obteniendo un error máximo en ciertas posiciones del 5%, lo que resulta satisfactorio.

Por último, en la función se añaden varios "plot" para graficar las circunferencias de piñón y corona, las líneas de cadena, la línea de squat y el punto de intersección de la línea de cadena con la extensión de la línea del basculante, con la idea de que aparezca en la app junto a la representación de la moto.

6.4. Dibujar_circuito

Uno de los pilares fundamentales del programa es la representación del trazado y el punto por el que está pasando el prototipo en el momento seleccionado, ya que nos indica dónde está ocurriendo lo que vemos en la gráfica de la moto. El AIM adquiere datos de latitud, longitud y altura, pero para poder graficar el circuito necesitamos pasar estos datos a coordenadas UTM.

La conversión de coordenadas geográficas (latitud, longitud y altura) a coordenadas UTM se realiza mediante una proyección cartográfica llamada **Proyección Transversa de Mercator**. Esta proyección transforma la superficie curva de la Tierra en un plano, permitiendo trabajar con distancias y áreas de forma más precisa.

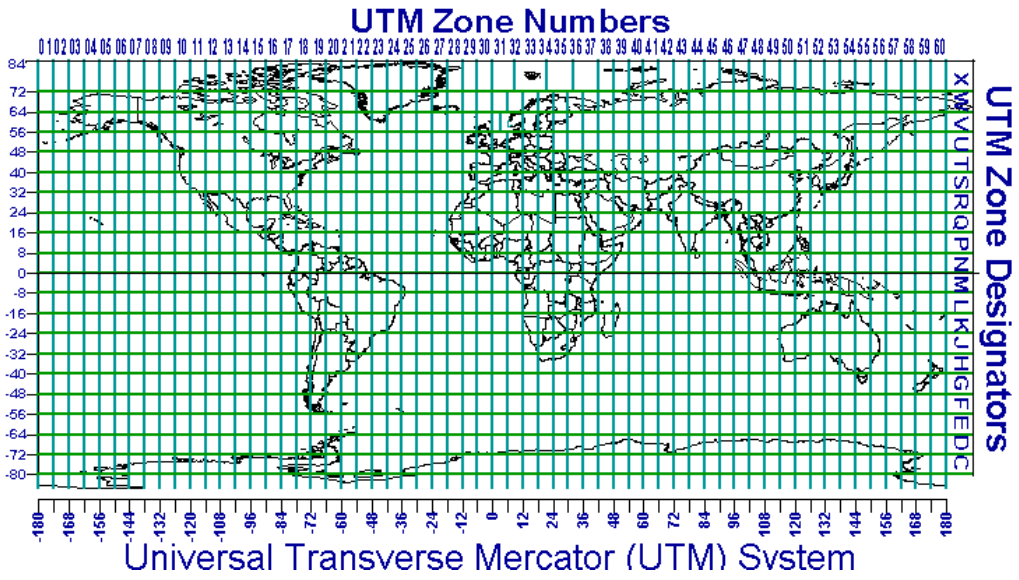

Figura 25 – Zonas UTM

- **División en zonas UTM**

El sistema UTM divide la Tierra en **60 zonas** de **6 grados de ancho** cada una, numeradas del 1 al 60 de oeste a este. Por ejemplo, España se encuentra mayoritariamente en las zonas 29, 30 y 31. Cada zona tiene un **meridiano central**, que sirve de referencia para los cálculos.

- **Sistema de referencia**

La conversión se hace con respecto a un **elipsoide de referencia**, normalmente el **WGS84**, que representa matemáticamente la forma de la Tierra.

- **Conversión**

Para transformar latitud y longitud en coordenadas UTM se aplican fórmulas matemáticas que tienen en cuenta:

- La curvatura de la Tierra
- La distancia al meridiano central de la zona
- Factores de escala para corregir deformaciones

El resultado son dos coordenadas planas:

- **Este (Easting)**: distancia desde el meridiano central de la zona, en metros. Se suma 500.000 m para evitar valores negativos.
- **Norte (Northing)**: distancia desde el ecuador, en metros. En el hemisferio sur se añade 10.000.000 m.

La **altura** (sobre el nivel del mar o sobre el elipsoide) normalmente se mantiene igual.

La función realiza esta conversión de latitud, longitud y altitud a coordenadas UTM. Como entrada tiene los ejes de la gráfica de la aplicación, latitud, longitud, altura, y posición en el vector de tiempo de los datos de telemetría para representar el punto de la moto encima del circuito.

6.5. Pintamoto

Esta es la aplicación de app designer de MATLAB. En esta sección se describirá el código que aborda la app.

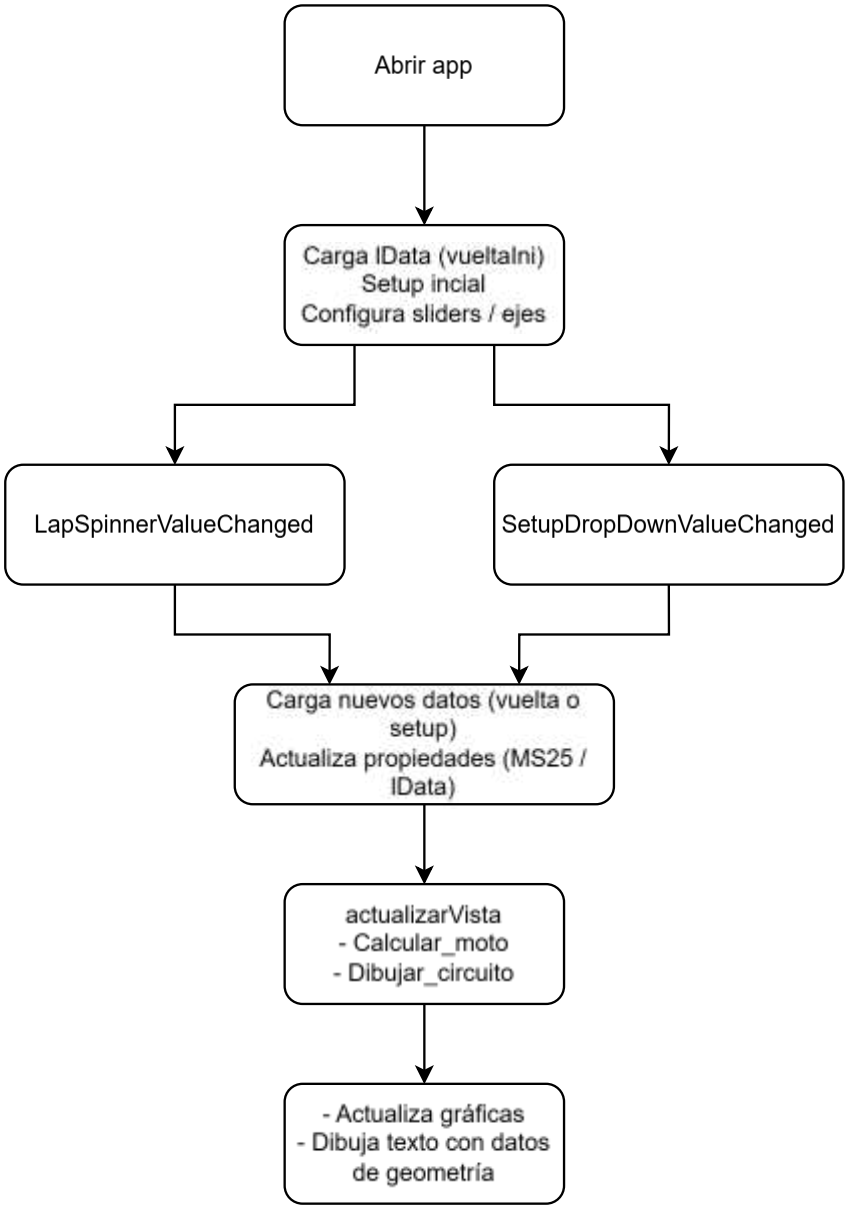

Figura 26 – Flujo de información dentro de la app

- `StartupFcn`: Dentro de la `startupFcn` se carga la estructura *lData* (que la usa todas las funciones dentro de la app) y los datos a estudiar con el nombre del archivo correspondiente. A continuación, un ejemplo de cómo se carga cada dato de *lData*:

$$app.lData.d = lData.\{1, vueltaIni\}.d$$

Y así, con todos los datos que contiene la estructura. Además, establecemos los límites del slider en función de la longitud de los datos cargados y se configuran los ejes de ambas gráficas, *UIAxes* (moto) y *UIAxes2* (circuito).

$$app.UIAxes.XTick = [\,];$$

$$app.UIAxes.YTick = [\,];$$

$$app.UIAxes2.XTick = [\,];$$

$$app.UIAxes2.YTick = [\,];$$

Por último, llama al `SetupDropDownValueChanged`, para cargar el setup inicial de la moto.

$$SetupDropDownValueChanged\ (app, [\,])$$

- `actualizarVista`: esta función se define dentro de la app dentro de un método de acceso privado. Esta, se encarga de la representación de la moto y el circuito en las respectivas gráficas destinadas para ello llamando a las funciones:

$$[\theta_{avance}, avance, AS_{\%}, R_{horquilla}, R_{trasero}, wheelbase]$$
$$= Calcular\_moto(app.MS25, app.lData.fs, app.lData.rs, value, app.UIAxes)$$

$$Dibujar\_circuito(app.lData.lat\ , app.lData.long, app.lData.alt, app.UIAxes2, value)$$

Hay que tener en cuenta que el parámetro value, en estas funciones se utiliza para dar el valor de la posición en el vector de datos. Este es un número entero, por tanto, antes de llamar a estas funciones, el parámetro value se redondea a un número entero:

$$value = round(app.Slider.value)$$

Y los datos de geometría se grafican encima de la gráfica de la moto, que también se actualiza en cada cambio del slider (ver anexo). En cada cambio de posición del slider se elimina la información mostrada en pantalla para que no se pise la información.

$$delete(findall(app.UIAxes, 'Type', 'text'))$$

Esta función es llamada en todas las demás funciones de los distintos elementos que conforman la app para su correcto funcionamiento.

- `SetupDropDownValueChanged`: esta función es la encargada de realizar el cambio entre las distintas geometrías introducidas en la aplicación. La función es definida automáticamente al posicionar en la app el `DropDown`, pero dentro de ella se escribe código. Lo que necesitamos es que, al cambiar de geometría, esto se vea reflejado instantáneamente en la app y que si no se encuentra el archivo deseado aparezca un mensaje de error en pantalla. Tener en cuenta que sea cual sea el archivo seleccionado, tiene que formar parte de la propiedad

MS25 definida al principio de la app. Para realizar el cambio entre setup se usa el comando switch:

```
switch app.SetupDropDown.Value
    case 'Aceleracion'
        archivo = 'MS25.mat';
        variable = 'MS25';
    case 'Carrera'
        archivo = 'MS25_carrera.mat';
        variable = 'MS25_carrera';
    case 'Jerez07/07'
        archivo = 'MS25_Jerez0707.mat';
        variable = 'MS25_Jerez0707';
    otherwise
        warning('Opción no reconocida: %s',
app.SetupDropDown.Value);
        return;
end
```

Puede parecer innecesario la parte de archivo y variable, ya que parecen lo mismo. Durante el desarrollo de la app y probando con distintos setups, vimos que es posible guardar en un mismo archivo, distintas estructuras, lo que generaba el problema de querer acceder a una estructura que se encontraba en un archivo con otro nombre, por tanto, aparecía un mensaje de error. Haciéndolo de esta manera, nos aseguramos de que independientemente de que sea un archivo distinto o dentro de un archivo una estructura distinta, cargamos la deseada.

Si se encuentra la estructura, carga la estructura y llama a `actualizarVista` para actualizar la información en pantalla. En caso contrario, aparece un mensaje de error en pantalla avisando de que no se pudo cargar el archivo en cuestión:

```
try
    datos = load(archivo, variable);   % ← Solo carga la variable
deseada
    app.MS25 = datos.(variable);      % ← Accede dinámicamente a la
variable cargada


    actualizarVista(app);
    SliderValueChanged(app, []);
catch ME
```

```
    errordlg(['No se pudo cargar el archivo: ', archivo], 'Error');
    disp(ME.message);
end
```

- `LapSpinnerValueChanged`: es la función encargada de cargar la vuelta seleccionada. En ella, como cambiamos de struct al cambiar de vuelta, se vuelve a cargar la estructura de datos completa y se llaman a los datos de la vuelta seleccionada:

$$app.lData.d = lData.\{1, app.LapSpinner.Value\}.d$$

Así con todos los datos. Y al igual que en la `startupFcn`, se configura la longitud del slider y los límites de este:

$$nData = length(app.lData.d)$$

$$app.Slider.Limits = [1, nData]$$

Como podemos ver, dentro de la app no hay mucho código, pero es el necesario para que funcione correctamente.

6.5.1. Ejemplos de uso de la app

En este apartado quiero mostrar unos ejemplos, simulando el caso de exportar los datos de la tanda y analizar ciertos puntos, en aras de mostrar la cantidad de conclusiones que se pueden extraer del programa.

- Frenada en la primera curva:

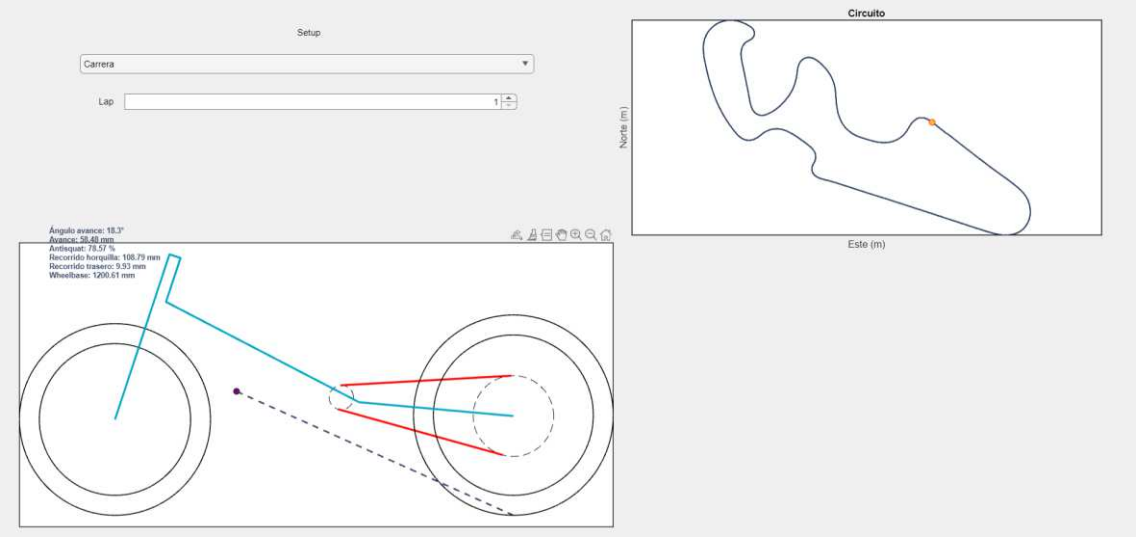

Figura 27 – Frenada en la primera curva de MotorLand

En esta vuelta, este es el punto de mayor frenada de toda la curva, como se puede deducir del recorrido de horquilla, 108,79 mm, teniendo en cuenta que la horquilla

tiene un recorrido total de 120 mm, se puede deducir que en este punto de la curva está muy cercana a hacer tope, pero quedando 10 mm de margen, por tanto, tiene una buena puesta a punto (analizando esta curva, queda sujeto a analizar todas las frenadas del circuito y de escuchar los comentarios del piloto). En este punto el dato de antisquat no es relevante, ya que solo afecta en aceleración y en este punto todavía está frenando preparando la entrada a curva. Vemos que el ángulo de avance de 18,3º es bastante cerrado, puede que el piloto note inestabilidad o el tren delantero muy nervioso en este punto, para eso están sus comentarios y, en tal caso, habría que estudiar si cambiar la resistencia del amortiguador de dirección o si subir la moto del tren delantero (bajando las barras de horquilla respecto a las tijas) para tener un mayor ángulo de avance. El recorrido trasero es muy bajo, algo lógico en frenada ya que se transfiere el peso al tren delantero, descomprimiendo el amortiguador trasero.

- Aceleración en la salida de la curva 15 (hacia la recta de atrás):

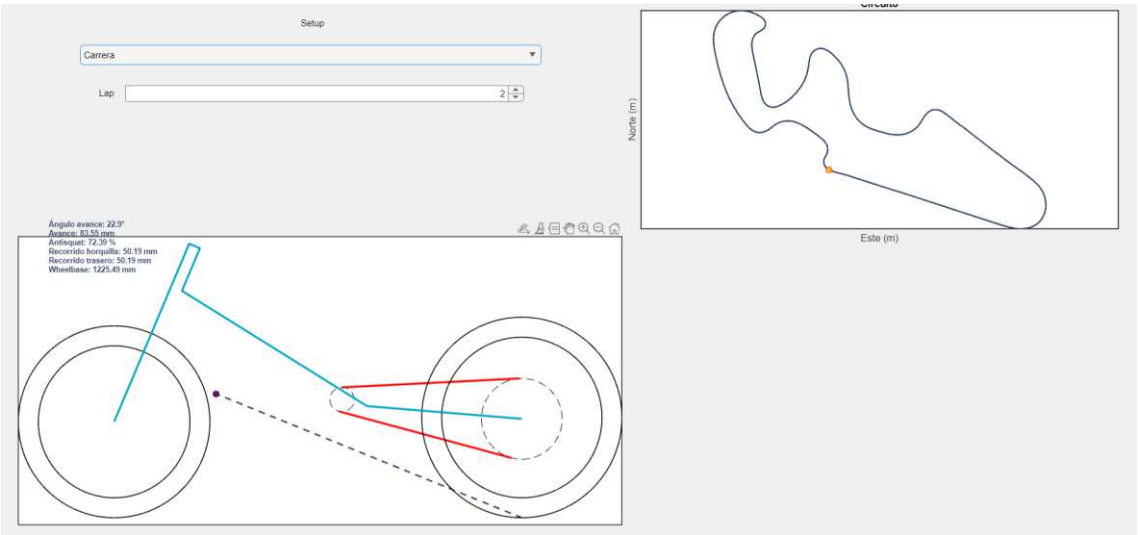

Figura 28 – Salida de la curva 15 en aceleración

Solo con ver la imagen, comparándola con el caso anterior, se puede ver cómo la moto está comprimida de la parte trasera y más extendida del tren delantero, que se puede verificar en los recorridos, ambos 50,19 mm. Al estar la horquilla menos comprimida, el ángulo de avance y el avance son mayores al caso anterior, de hecho, mayor ángulo de avance al de la moto completamente extendida (22,5 º).

Por último, el antisquat, que en este caso si es importante ya que nos encontramos en una aceleración. Tenemos un valor de 72,39 %, lo que nos indica que al acelerar la moto va a comprimir en amortiguador en cierta cantidad. El piloto es muy sensible a cambios del antisquat, ya que le cambia la percepción de la “dureza” del tren trasero. Imaginemos que el piloto nos comenta que nota que la moto al acelerar se

comprime demasiado, pero que las imperfecciones de la pista las lee correctamente el tren trasero. Esto nos indica que, el muelle de amortiguador es el correcto, pero que el antisquat es demasiado bajo para el gusto del piloto y sus sensaciones. Habrían varias opciones para cambiar el valor de antisquat como cambiar las alturas (cambia altura del centro de gravedad), cambiar la posición del pivot (eje del basculante, ajuste habitual en motos avanzadas tecnológicamente como las Superbikes o MotoGP), o cambiar la relación de transmisión mediante el piñón y la corona.

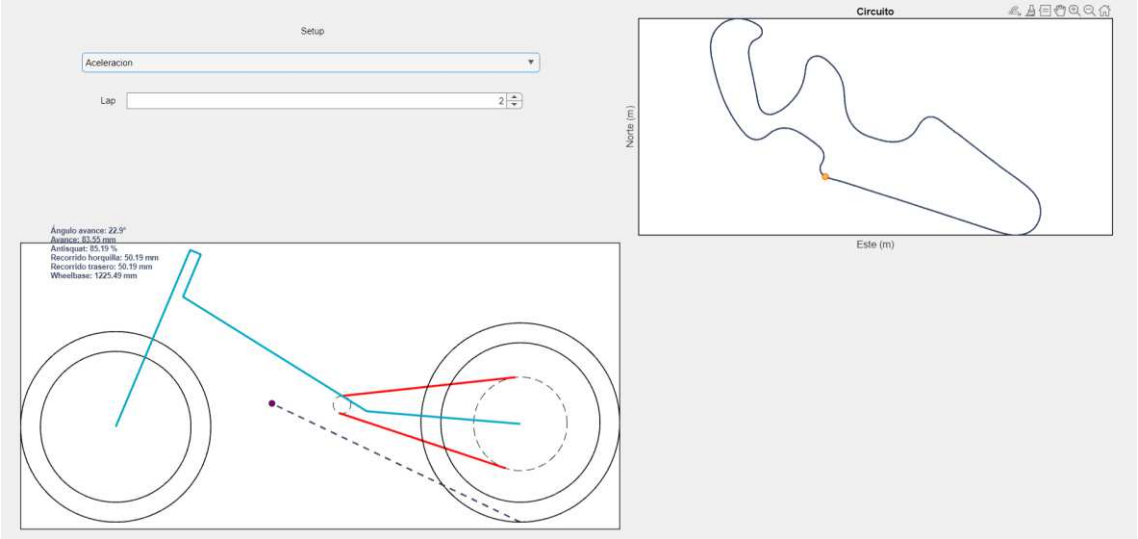

Figura 29 – Cambio al setup de Aceleración

Se ha cambiado al setup de aceleración, que tiene otra relación de transmisión y el pivot subido 10 mm respecto al setup de carrera. Vemos que el valor de antisquat ha aumentado algo más de un 10%. Sería una posible solución.

- Punto medio de la última curva:

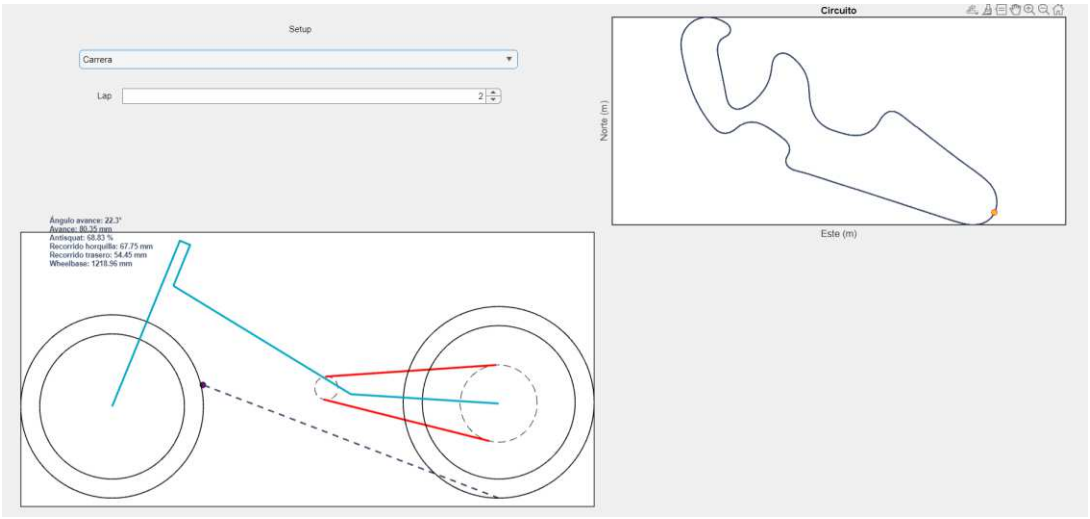

Figura 30 – Punto medio de la última curva

En este caso, estamos estudiando la última curva que tiene un radio muy grande y en estas motos se toman a gran velocidad, por tanto, es un punto crítico en cuanto a sensaciones del piloto y puesta a punto, se puede ganar o perder mucho tiempo en esta curva, el piloto se tiene que sentir lo más cómodo posible y con agarre de los neumáticos. El valor de antisquat que nos arroja en este caso, es un valor relativamente bajo pero que, en este punto de la curva, un valor alto cargaría más el neumático trasero, haciendo que soporte mayores esfuerzos, aumentando la temperatura. La moto se ve bastante nivelada en cuanto a suspensiones, lo que indica una buena puesta a punto en este apartado.

7. Manual de usuario

Este apartado sirve como manual para el correcto uso de la app por cualquier persona. A continuación, se explica paso a paso cómo usar la app y qué cambiar de las distintas funciones que componen la app:

- Exportar datos de Race Studio a MATLAB: abrimos Race Studio y seleccionamos la tanda que queremos exportar. Una vez nos encontramos en la pestaña de análisis de datos nos vamos a la pestaña: Archivo > Exportar datos

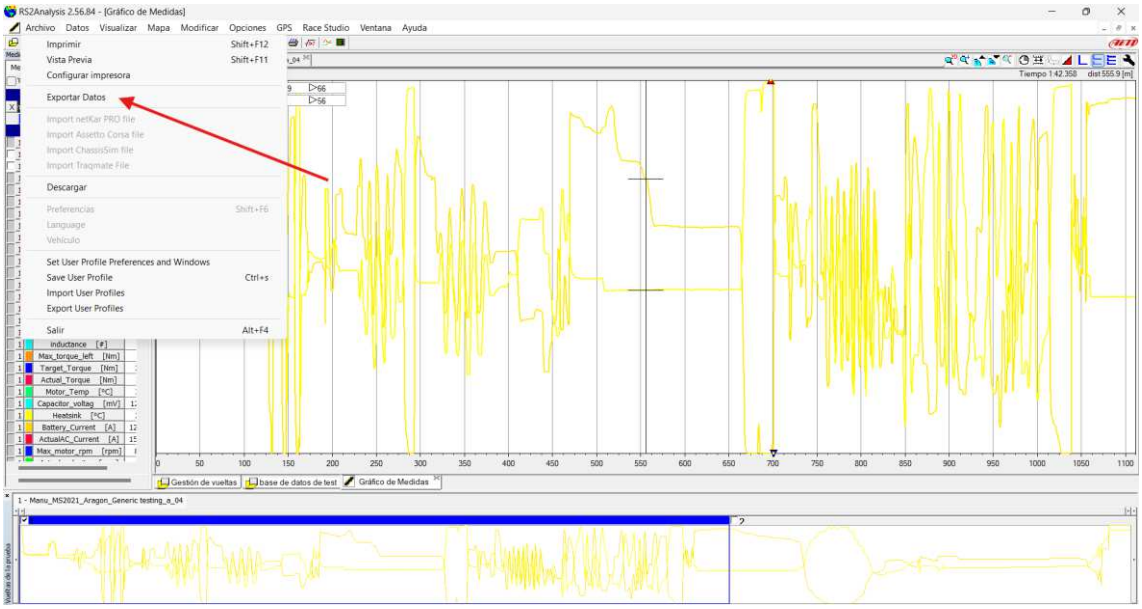

Figura 31 – Pestaña para exportar datos

Y una vez nos encontramos en la pestaña “data export” seleccionamos los canales de datos a exportar, que en nuestro caso son los recorridos de suspensión delantera y trasera, latitud, longitud y altura. Seleccionamos las vueltas a exportar, archivo MATLAB, guardar y salir.

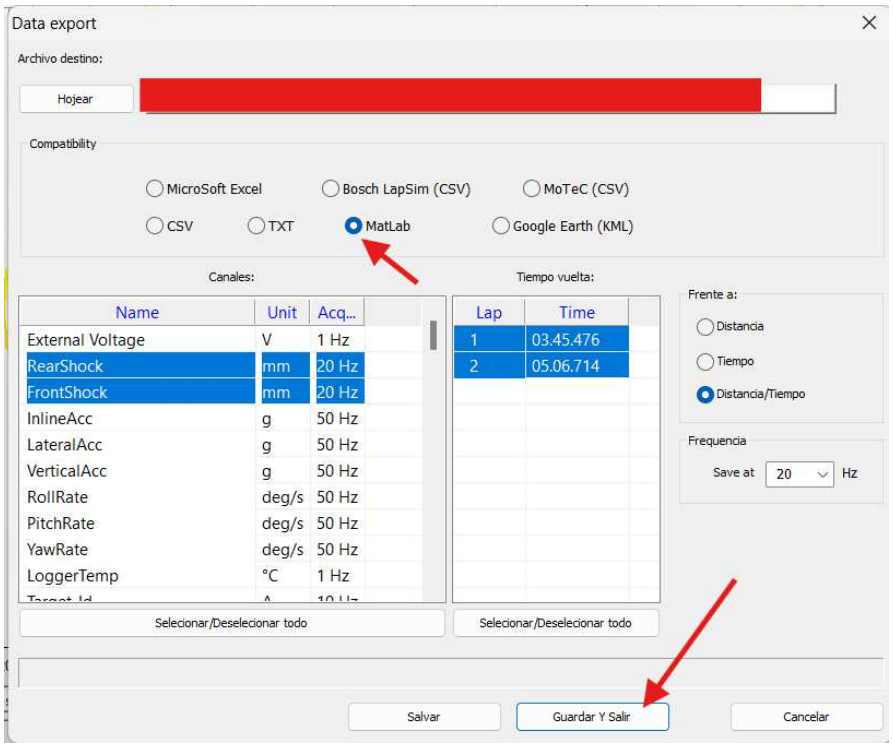

Figura 32 – Data export

Importante guardarlo en la carpeta en la que estemos trabajando en MATLAB (Current folder).

- Cargar datos en la estructura *lData* para que sean compatibles con la app: necesitamos el archivo exportado de Race Studio, pasarlo a la estructura *lData*, con la que trabaja la app, en formato `.mat`. Abrimos la función `convertirDatosTelemEstructura` y en la línea 2 le ponemos al load el archivo que hemos exportado en el primer paso, con su nombre. Acto seguido, en la línea 4, cambiamos el for en función de las vueltas exportadas. Por ejemplo, si hemos exportado las vueltas dos, tres y cuatro de una tanda, dentro del for pondríamos i = 2:4. No podemos exportar vueltas no consecutivas, al no ser que sea de forma individual. En las líneas 13 y 17 ponemos el nombre que queramos a la estructura de datos. Ejecutamos y ya tendremos la estructura perfectamente en la carpeta con la que estemos trabajando en MATLAB.

```
clear; close all; clc;
load    NombreDelArchivoExportadoDeRS2.mat;    %Cargamos    archivo
proveniente de RaceStudio


for i = 2:3 %Modificamos para el número de vueltas que exportamos
    lData{i}.d  = eval(sprintf('L00%i_distance',i));
    lData{i}.fs = eval(sprintf('L00%i_FrontShock',i));
```

```
    lData{i}.rs = eval(sprintf('L00%i_RearShock',i));
    lData{i}.t = eval(sprintf('L00%i_time',i));
    lData{i}.lat = eval(sprintf('L00%i_latitude',i));
    lData{i}.long = eval(sprintf('L00%i_longitude',i));
    lData{i}.alt = eval(sprintf('L00%i_GPS_Altitude',i));
    lData{i}.RollRate=eval(sprintf('L00%i_RollRate',i));
    lData{i}.LateralAcc=eval(sprintf('L00%i_LateralAcc',i));
    lData{i}.VerticalAcc=eval(sprintf('L00%i_VerticalAcc',i));
end
save NombreEstructura.mat lData; %Lo nombramos como queramos

clear;

    load NombreEstructura.mat
```

- En el código de la app, cambiar el load de las líneas 56 y 94, y poner el nombre de archivo que hemos creado en el apartado anterior, u otro si ya lo tenemos.
- En la línea 57, cambiar *vueltaIni* a la vuelta inicial de los archivos que hemos cargado. De la misma forma, cambiar los límites del *LapSpinner* en el "design view", para evitar problemas al ejecutar la app en el caso de que seleccionemos una vuelta no exportada.

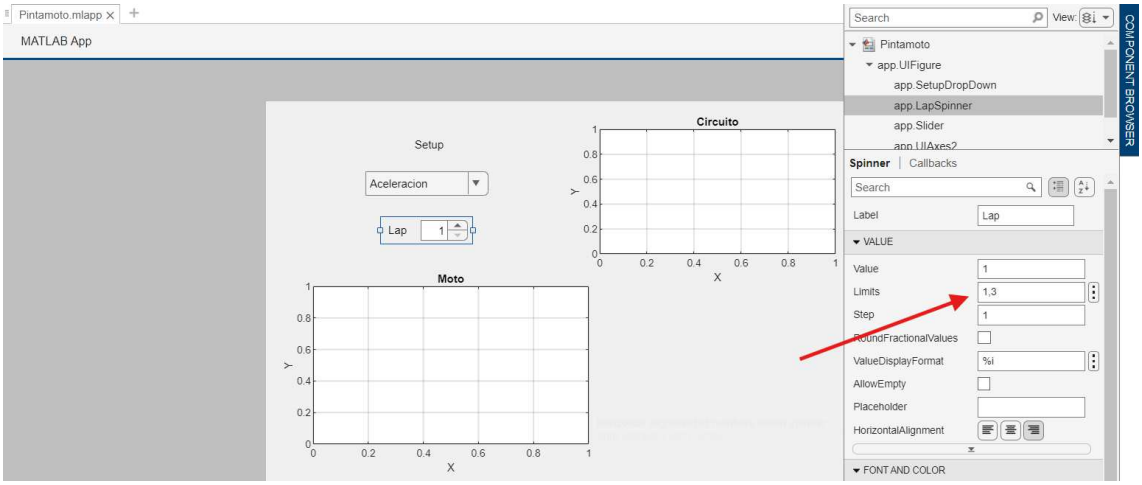

Figura 33 – Cambiar límites al LapSpinner

- En la función `SetupDropDownValueChanged`, en los casos de aceleración, carrera, etc, cargar la estructura que contenga la geometría de moto que queremos estudiar:

```
switch app.SetupDropDown.Value
```

```
    case 'Aceleracion'
        archivo = 'MS25.mat';
        variable = 'MS25';
    case 'Carrera'
        archivo = 'MS25_carrera.mat';
        variable = 'MS25_carrera';
    case 'Jerez07/07'
        archivo = 'MS25_Jerez0707.mat';
        variable = 'MS25_Jerez0707';
```

Incluso podríamos añadir otro case. Por ejemplo, si hemos ido a una tanda y queremos añadir la moto `Vmax.mat`, y no queremos modificar ninguna de las que hay añadidas, después del último case añadimos otro:

```
case 'Vmax'
        archivo = 'Vmax.mat';
        variable = 'Vmax';
```

Y también tenemos que cambiar en el "code view", en el Drop Down hay que modificar el nombre en los ítems

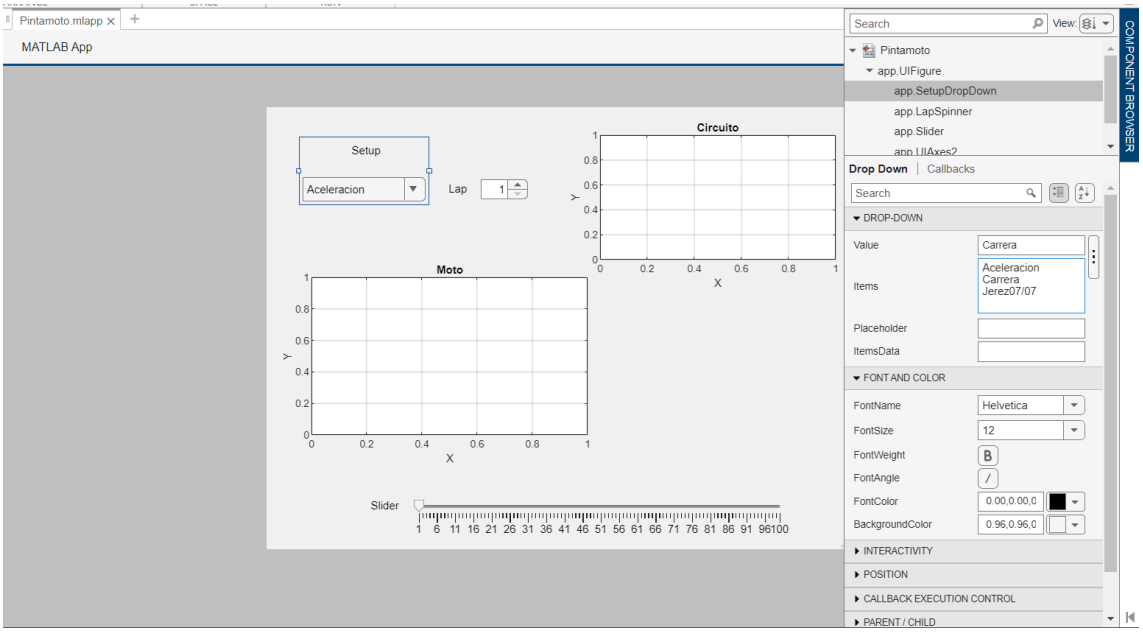

Figura 34 – Cambio de nombre del Drop Down

Ya nos aparecería al ejecutar la app en el desplegable de las distintas configuraciones.

- Ejecutar app.

- Dentro de la app nos encontramos el desplegable, con el que podemos cambiar de configuraciones de moto.

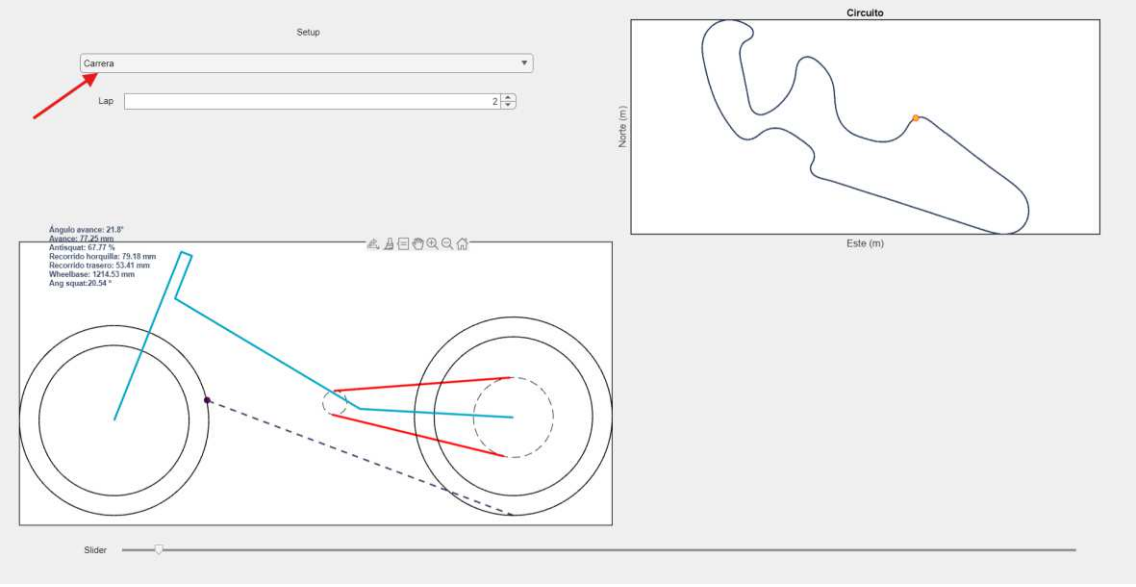

Figura 35 – Selector de geometría

Para ver los datos de forma real, tenemos que poner la configuración con la que hemos salido a pista, para que la geometría coincida con la usada. Las demás configuraciones se pueden usar de comparación para ver los parámetros geométricos, incluso se puede añadir una configuración que realice un cambio en la geometría que pensemos que puede ser válido y verificarlo con los datos de geometría de la vuelta en cuestión. Encontramos un selector de vuelta para cambiar entre las distintas vueltas que hemos exportado.

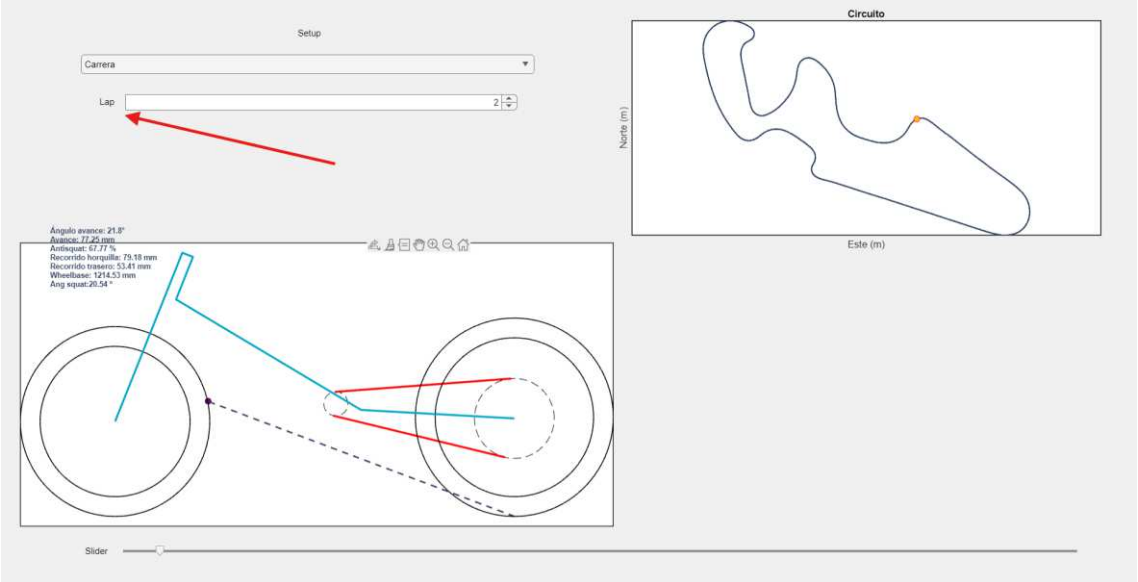

Figura 36 – Selector de vuelta

En el caso de seleccionar una vuelta no exportada, saldrá un mensaje de error en la pantalla al no encontrar datos de esa vuelta elegida. Para evitar esto, se sugiere cambiar los límites de LapSpinner.

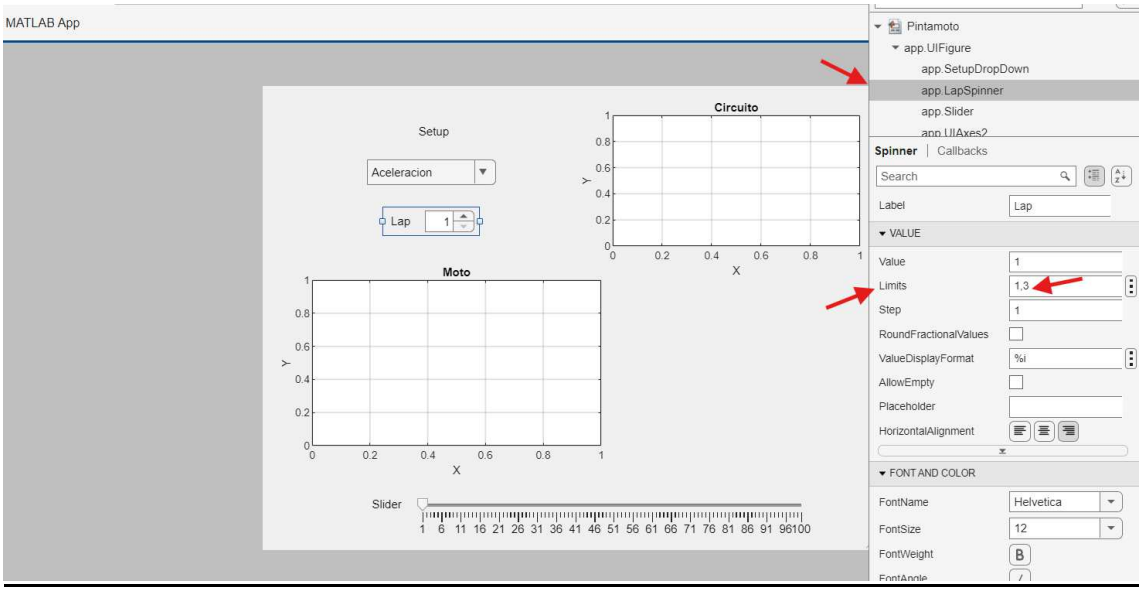

Figura 37 – Cambiar límite del LapSpinner

El slider inferior sirve para avanzar en la vuelta. Empieza en el comienzo de la vuelta y acaba en el final, por tanto, podemos recorrer todo el circuito con el slider que se ajusta a la longitud de la vuelta.

8. Conclusiones

Una vez finalizado el proyecto, se pueden sacar conclusiones con claridad del trabajo realizado.

El objetivo principal era realizar una plataforma interactiva fácilmente usable por cualquiera independientemente del nivel de conocimiento del usuario y se ha conseguido.

La aplicación es fácil de usar, muestra los datos que consideramos de mayor importancia y, con el manual de usuario, puede usarla cualquier persona que tenga acceso a MATLAB en futuras ediciones de MotoStudent. Es una herramienta de gran ayuda tanto para usuarios con poco conocimiento, que pueden aprender a interpretar datos con la aplicación, tanto para usuarios experimentados en analizar datos, ya que, de un vistazo rápido, se puede ver la posición de la moto y parámetros de importancia en la puesta a punto.

Otro de sus puntos a favor es el posible estudio de distintas puestas a punto. Al poder incluir distintos set-ups, pueden intercambiarse indistintamente y, de esta manera, ver qué geometría nos puede interesar más en todos los puntos de la vuelta que estemos estudiando.

Además, la aplicación no reemplaza a otros programas de análisis de datos más tradicionales, más bien es otra herramienta con un enfoque distinto, que puede usarse en paralelo o individualmente.

Es una herramienta más de la que dispone UMA Racing Team para seguir mejorando, innovar y mantenerse en lo más alto de la competición.

9. Líneas futuras

El presente documento, como ya se comentó en las conclusiones, trató de realizar un programa completo, pero con la información que consideramos más importante.

Aumentar la complejidad del programa añadiendo más datos (corrientes eléctricas, tensiones, temperatura de agua) es una opción, aunque perdería la esencia de la simplicidad de uso del programa actual.

Sería interesante intentar implementarlo a un modelo multicuerpo del prototipo. En el UMA Racing Team, tienen experiencia con los modelos multicuerpo y tenemos uno del prototipo del 2023. Con el modelo del último prototipo, sería interesante trasladar esta idea al modelo en tres dimensiones y estudiar las opciones que se pudieran implementar en SimScape.

10. Referencias

[1] C. Morante, «Curso teórico de puesta a punto - Parte ciclo,» 2022.

[2] C. Morante, «Curso Teórico de puesta a Punto - Telemetría,» 2019.

[3] T. Foale, Motorcycle Handling and Chassis Design the art and science, Tony Foale, 2002.

[4] V. Cossalter, R. Lot and M. Massaro, Motorcycle Dynamics, 2014, pp. 1-42.

[5] V. Cossalter y R. Lot, «A Motorcycle Multi-Body Model for Real Time Simulations Based on the Natural Coordinates Approach,» 2002.

[6] . Mathworks, Simscape Multibody User's Guide, 2024.

[7] M. Guiggiani, The Science of Vehicle Dynamics: Handling, Braking, and Ride of Road and Race Cars, 2 ed., Springer International Publishing, 2018, pp. 1-550.

[8] F. P. Beer, E. R. Johnston y P. J. Cornwell, Mecánica vectorial para ingenieros - Dinámica, 9 ed., McGraw-Hill, 2010.

[10] C. O. Wilke, Fundamentals of Data Visualization, 1 ed., O'Reilly Media, 2019.

[11] P. Thede y L. Parks, Race Tech's motorcycle suspension bible, Motorbooks, 2010, p. 254.

[12] D. Croccolo y M. De Agostinis, Motorbike Suspensions Modern Design and Optimisation, Springer, 2013.

[13] C. Motorsport, Chasis Program Manual v24.0.

[14] A. Sportline, Race Studio 2 Analysis User´s Guide, 2003.

ANEXOS

- Calcular_moto

```
function [angulo_avance, avance, AS_percent, Recorrido_horquilla,
Recorrido_trasero,wheelbase,squat_angle] = Calcular_moto(bike, Ft,
Rt, i, ax)
    cla(ax);
    axis(ax, 'equal');


     % Ajuste por el modo en que mide el sensor de shock trasero
     Rt(i,1)=max(Rt)-Rt(i,1);
     Rt(i, 1) = 281 - Rt(i, 1);


     %Regresión lineal del ángulo chasis-basculante frente al
recorrido de
     %amortiguador
    ch_sw_angle = 3.239e-06 * Rt(i, 1)^3 - 0.002502 * Rt(i, 1)^2 +
0.8811 * Rt(i, 1) + 35.66;


    % Recorridos
    Ft(i,1)=120-Ft(i,1);
    Recorrido_horquilla = Ft(i,1);
    Recorrido_trasero = -1.933 * Rt(i,1) + 553.1; % Regresión
lineal, pares de puntos obtenidos en SolidWorks


    % Coordenadas de la moto
    x_coor = [0, ...
        (bike.fork_length - Ft(i, 1)) * cosd(90), ...
        (bike.fork_length - Ft(i, 1)) * cosd(90) +
bike.yoke_offset, ...
        (bike.fork_length - Ft(i, 1)) * cosd(90) + bike.yoke_offset
+ bike.steam_to_upyoke * cosd(-90), ...
        (bike.fork_length - Ft(i, 1)) * cosd(90) + bike.yoke_offset
+ bike.steam_to_upyoke * cosd(-90) + bike.pivot_to_steam * cosd(-
9.26), ...
        (bike.fork_length - Ft(i, 1)) * cosd(90) + bike.yoke_offset
+ bike.steam_to_upyoke * cosd(-90) + bike.pivot_to_steam * cosd(-
9.26) + bike.swingarm * cosd(180 - ch_sw_angle - 9.26)];


    y_coor = [0, ...
        (bike.fork_length - Ft(i, 1)) * sind(90), ...
        (bike.fork_length - Ft(i, 1)) * sind(90), ...
        (bike.fork_length - Ft(i, 1)) * sind(90) +
bike.steam_to_upyoke * sind(-90), ...
```

```
        (bike.fork_length - Ft(i, 1)) * sind(90) +
bike.steam_to_upyoke * sind(-90) + bike.pivot_to_steam * sind(-
9.26), ...
        (bike.fork_length - Ft(i, 1)) * sind(90) +
bike.steam_to_upyoke * sind(-90) + bike.pivot_to_steam * sind(-
9.26) + bike.swingarm * sind(180 - ch_sw_angle - 9.26)];


    %Coordenadas del centro de gravedad y eje motor, trasladan y
giran
    %respecto al chasis
    x_cdg_pn=[(bike.fork_length - Ft(i, 1)) * cosd(90) +
463.28*cosd(5.49), (bike.fork_length - Ft(i, 1)) * cosd(90) + 662.17
* cosd(337.47)];
    y_cdg_pn=[(bike.fork_length - Ft(i, 1)) * sind(90) +
463.28*sind(5.49), (bike.fork_length - Ft(i, 1)) * sind(90) + 662.17
* sind(337.47)];


    %Cálculos de ángulos de giro para matriz de rotación
    wheelbase = sqrt(x_coor(6)^2 + y_coor(6)^2);
    th_diff_wheel = atand((bike.rear_wheel - bike.front_wheel) /
wheelbase);
    th_change = atand(y_coor(6) / x_coor(6));


    Coor = [x_coor; y_coor];
    R = [cosd(-th_change + th_diff_wheel), -sind(-th_change +
th_diff_wheel); ...
         sind(-th_change + th_diff_wheel), cosd(th_change +
th_diff_wheel)];


    %Coordenadas giradas para representar de forma correcta la moto
    Coor_giradas = R * Coor;
   Coor_giradas(2, :) = bike.front_wheel + Coor_giradas(2, :);


   %Lo mismo para las coordenadas del cdg y eje motor
     cdg_pn=[x_cdg_pn; y_cdg_pn];
     cdg_pn=R*cdg_pn;
     cdg_pn(1,:)=cdg_pn(1,:)-Coor_giradas(1,6);
     cdg_pn(2,:)=bike.front_wheel+cdg_pn(2,:);


    % Ángulo de avance y avance
    angulo_avance = atand((Coor_giradas(1, 2) - Coor_giradas(1, 1))
/ ...
                          (Coor_giradas(2, 2) - Coor_giradas(2,
1)));
```

```
    x = bike.yoke_offset / cosd(angulo_avance);
    y = x / tand(angulo_avance);
    z = (bike.front_wheel - y) / cosd(angulo_avance);
    avance = sqrt(z^2 - (bike.front_wheel - y)^2);
    wheelbase=Coor_giradas(1,6)-Coor_giradas(1,1)

    %Llamada a función Antisquat

[squat_angle,load_transfer_angle,AS_percent]=Antisquat(cdg_pn(2,1)
,cdg_pn(2,2),Coor_giradas(1,5),Coor_giradas(2,5),Coor_giradas(1,6)
,Coor_giradas(2,6),cdg_pn(2,1),bike.pinon,bike.corona,wheelbase,ax
)

    % Dibujar ruedas y chasis
    wheel(ax, 0, bike.front_wheel, bike.front_wheel);
    wheel(ax, 0, bike.front_wheel, bike.front_wheel - 60);
    wheel(ax,      Coor_giradas(1,      6),      bike.rear_wheel,
bike.rear_wheel);
    wheel(ax, Coor_giradas(1, 6), bike.rear_wheel, bike.rear_wheel
- 60);

    chasis = line(ax, Coor_giradas(1, :), Coor_giradas(2, :), ...
        'Color', '#00A5C5', 'LineWidth', 2);

    drawnow;
end
```

- Antisquat

```
function
[squat_angle,load_transfer_angle,AS_percent]=Antisquat(x_pinon,y_p
inon,x_pv,y_pv,x_rw,y_rw,y_cdg,zp,zc,wheelbase,ax)


    p_chain = 12.7;%*1/(8*25.4); %12.7 es el paso para cadenas 415


    a1 = x_pinon-wheelbase; %Coordenada x de piñon
    b1 = y_pinon; %Coordenada y del piñon
    r_pinon = p_chain/sind(180/zp)/2; %Radio primitivo del piñon


    a2 = 0; %Coordenada x de rueda trasera
    b2 = y_rw; %Coordenada y de rueda trasera
    r_corona = p_chain/sind(180/zc)/2; % Radio primitivo de la
corona


    a21 = a2-a1; b21 = b2-b1; %distancias x e y entre eje trasero
y piñon
    d2 = a21.^2+b21.^2; %distancia de eje trasero a piñon
    r21 = (r_corona-r_pinon)./d2; %diferencia de radios entre linea
de eje trasero a piñon
    s21 = sqrt(d2-(r_corona-r_pinon)^2)./d2;
    u1 = [-a21.*r21-b21.*s21;-b21.*r21+a21.*s21]; %Vector unitario
de la cadena superior
    u2 = [-a21.*r21+b21.*s21;-b21.*r21-a21.*s21];
    L1 = [a1',b1']+r_pinon*u1'; %Punto de tangencia de la cadena en
el piñon
    L2 = [a2',b2']+r_corona*u1'; %Punto de tangencia de la cadena
en la corona
    R1 = [a1',b1']+r_pinon*u2'; R2 = [a2',b2']+r_corona*u2';
%Puntos de tangencia de la rama inferior de cadena


    % Verifica la dirección real de la cadena
    vector_cadena = L2 - L1;
        if vector_cadena(1) < 0   % la cadena apunta hacia la
izquierda: mal
        % Invertimos dirección de u1
        u1 = -u1;


        % Recalculamos puntos de tangencia
        L1 = [a1, b1] + r_pinon * u1';
        L2 = [a2, b2] + r_corona * u1';
        end
```

```
    % Ecuación de la cadena
    m_chain = (L1(:,2)-L2(:,2))./(L1(:,1)-L2(:,1)); %Pendiente de
la cadena
    n_chain = L2(:,2)-m_chain.*L2(:,1); %Ordenada en el origen de
la cadena n = y - m*x


    % Ecuación del basculante
    sw_theta=atan2d((y_pv-y_rw),(x_pv-x_rw));
    m_sw = tand(sw_theta)'; %Pendiente del basculante
    n_sw = y_rw; %Ordenada en el origen del basculante


    x_squat = (n_chain-n_sw)./(m_sw-m_chain); %Coordenada x del
punto de interseccion entre linea de cadena y basculante
    y_squat = m_sw.*x_squat+n_sw; %Coordenada y del punto de
interseccion entre linea de cadena y basculante


    % Ángulo de squat
    squat_angle = atand(y_squat./-x_squat);


    % Antisquat ratio
    load_transfer_angle = atan2d(y_cdg,wheelbase);
    AS_percent = tand(squat_angle)./tand(load_transfer_angle)*100;


       % === PLOTEO VISUAL DE COMPONENTES DE CADENA Y LINEA DE SQUAT
===
    hold(ax, 'on');


    % Dibujar circunferencias del piñón y corona
    theta = linspace(0, 2*pi, 100);
    plot(ax, x_pinon + r_pinon*cos(theta), y_pinon +
r_pinon*sin(theta), '--k');
    plot(ax, x_rw + r_corona*cos(theta), y_rw +
r_corona*sin(theta), '--k');


    % Dibujar línea de la cadena (tangente)
    plot(ax, [wheelbase+L1(1), wheelbase+L2(1)], [L1(2), L2(2)],
'r', 'LineWidth', 2);
    plot(ax, [wheelbase+R1(1), wheelbase+R2(1)], [R1(2), R2(2)],
'r', 'LineWidth', 2);


    % Dibujar punto de intersección (squat)
```

```
    plot(ax, (wheelbase+x_squat), y_squat, 'ko', 'MarkerFaceColor',
'#800080', 'MarkerSize', 6);


    % Dibujar línea de squat
    plot(ax, [wheelbase, (wheelbase+x_squat)], [0, y_squat], '--',
'Color', '#2A3B5C', 'LineWidth', 1.5);


    hold(ax, 'off');
```

- Dibujar_circuito

```
function Dibujar_circuito(lat, long, alt, ax, value)
    cla(ax);
    axis(ax, 'equal');

    % Convertir latitud y longitud a coordenadas UTM
    [E, N, ~, ~] = gps2utm(lat, long, alt);

    % Dibujar el circuito
    plot(ax, E, N, 'k', 'LineWidth', 1.5,'Color','#2A3B5C');
    hold(ax, 'on');

    % Verificar que "value" esté en el rango correcto
    if value < 1
        value = 1;
    elseif value > length(E)
        value = length(E);
    end

    % Obtener la posición UTM del punto en "value"
    E_punto = E(value);
    N_punto = N(value);

    % Dibujar el punto en la gráfica
    plot(ax, E_punto, N_punto, 'ro', 'MarkerSize', 6,
'MarkerFaceColor', '#FAB92C');

    % Etiquetas y título
    xlabel(ax, 'Este (m)');
    ylabel(ax, 'Norte (m)');
    title(ax, 'Circuito');
    grid(ax, 'on');

    hold(ax, 'off');
end

function [E, N, zone, H] = gps2utm(lat, lon, alt)
    % gps2utm convierte coordenadas de latitud, longitud y altura
(en grados y metros) a coordenadas UTM
% Entrada:
    %   lat: Latitud en grados
```

```
    %   lon: Longitud en grados
    %   alt: Altura en metros
    % Salida:
    %   E: Coordenada Este (Easting)
    %   N: Coordenada Norte (Northing)
    %   zone: Zona UTM
    %   H: Altura (misma que la de entrada)

    % Crear proyección UTM, utilizando el elipsoide WGS84
    utmstruct = defaultm('utm');

    % Determinar la zona UTM en función de la longitud
    zone = floor((lon(1) + 180) / 6) + 1;
    if lat(1) < 0
        utmstruct.zone = [num2str(zone) 'S'];
    else
        utmstruct.zone = [num2str(zone) 'N'];
    end

    % Definir el elipsoide WGS84
    utmstruct.geoid = wgs84Ellipsoid();
    utmstruct = defaultm(utmstruct);

    % Proyección hacia coordenadas UTM
    [E, N] = projfwd(utmstruct, lat, lon);

    % Altura permanece la misma
    H = alt;
end
```

- convertirDatosTelemEstructura

```
clear; close all; clc;
load Prueba13052025.mat; %Cargamos archivo proveniente de RaceStudio

for i = 2:3 %Modificamos para el número de vueltas que exportamos
    lData{i}.d  = eval(sprintf('L00%i_distance',i));
    lData{i}.fs = eval(sprintf('L00%i_FrontShock',i));
    lData{i}.rs = eval(sprintf('L00%i_RearShock',i));
    lData{i}.t = eval(sprintf('L00%i_time',i));
    lData{i}.lat = eval(sprintf('L00%i_latitude',i));
    lData{i}.long = eval(sprintf('L00%i_longitude',i));
    lData{i}.alt = eval(sprintf('L00%i_GPS_Altitude',i));
    lData{i}.RollRate=eval(sprintf('L00%i_RollRate',i));
    lData{i}.LateralAcc=eval(sprintf('L00%i_LateralAcc',i));
    lData{i}.VerticalAcc=eval(sprintf('L00%i_VerticalAcc',i));
end
save estructura4.mat lData; %Lo nombramos como queramos

clear;

load estructura4.mat
```

- Pintamoto

```
classdef Pintamoto < matlab.apps.AppBase


    % Properties that correspond to app components
    properties (Access = public)
        UIFigure            matlab.ui.Figure
        SetupDropDown       matlab.ui.control.DropDown
        SetupDropDownLabel  matlab.ui.control.Label
        LapSpinner          matlab.ui.control.Spinner
        LapSpinnerLabel     matlab.ui.control.Label
        Slider              matlab.ui.control.Slider
        SliderLabel         matlab.ui.control.Label
        UIAxes2             matlab.ui.control.UIAxes
        UIAxes              matlab.ui.control.UIAxes
    end


    properties (Access = public)
        lData; % Description
        MS25;
    end


    methods (Access = private)


       function actualizarVista(app)
    value = round(app.Slider.Value);


    [angulo_avance,
avance,AS_percent,Recorrido_horquilla,Recorrido_trasero,wheelbase,
~] = Calcular_moto(app.MS25, app.lData.fs , app.lData.rs, value,
app.UIAxes);


    Dibujar_circuito(app.lData.lat, app.lData.long, app.lData.alt,
app.UIAxes2, value);


    delete(findall(app.UIAxes, 'Type', 'text'));


    infoStr = sprintf(['Ángulo avance: %.1f°\n' ...
                   'Avance: %.2f mm\n' ...
                   'Antisquat: %.2f %%\n' ...
                   'Recorrido horquilla: %.2f mm\n' ...
                   'Recorrido trasero: %.2f mm\n' ...
```

```
                    'Wheelbase: %.2f mm\n' ...
                    ], ...
                    angulo_avance,        avance,        AS_percent,
Recorrido_horquilla, Recorrido_trasero,wheelbase);

    text(app.UIAxes, 0.05, 0.95, infoStr, ...
        'Units', 'normalized', ...
        'Color', '#2A3B5C ', ...
        'FontSize', 10, ...
        'FontWeight', 'bold');
       end
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
    load estructura4.mat lData;
    vueltaIni = 2;
    app.lData.d = lData{1,vueltaIni}.d;
    app.lData.fs = lData{1,vueltaIni}.fs;
    app.lData.rs = lData{1,vueltaIni}.rs;
    app.lData.t = lData{1,vueltaIni}.t;
    app.lData.lat = lData{1,vueltaIni}.lat;
    app.lData.long = lData{1,vueltaIni}.long;
    app.lData.alt = lData{1,vueltaIni}.alt;

    % Configuración visual
    axis(app.UIAxes, 'equal');
    app.UIAxes.XTick = [];
    app.UIAxes.YTick = [];
    app.UIAxes2.XTick = [];
    app.UIAxes2.YTick = [];
    app.Slider.MajorTicks = [];
    app.Slider.MinorTicks = [];

    % Ajustar los límites del slider
    nData = length(app.lData.d);
```

```
    app.Slider.Limits = [1, nData];

    % Cargar el archivo adecuado según el valor del DropDown
    SetupDropDownValueChanged(app, []);
        end

        % Value changed function: Slider
        function SliderValueChanged(app, event)
             actualizarVista(app);
        end

        % Value changed function: LapSpinner
        function LapSpinnerValueChanged(app, event)
            j = app.LapSpinner.Value;
            load estructura4.mat lData;
            app.lData.d = lData{1,j}.d;%Yo le añadi el ,j
            app.lData.fs = lData{1,j}.fs;
            app.lData.rs = lData{1,j}.rs;
            app.lData.t = lData{1,j}.t;
            app.lData.lat = lData{1,j}.lat;
            app.lData.long = lData{1,j}.long;
            app.lData.alt = lData{1,j}.alt;

            nData = length(app.lData.d);    % Numero de datos de la
vuelta j
            app.Slider.Limits = [1,nData];

            actualizarVista(app);
        end

        % Value changed function: SetupDropDown
        function SetupDropDownValueChanged(app, event)
switch app.SetupDropDown.Value
    case 'Aceleracion'
        archivo = 'MS25.mat';
        variable = 'MS25';
    case 'Carrera'
        archivo = 'MS25_carrera.mat';
        variable = 'MS25_carrera';
    case 'Jerez07/07'
```

```
            archivo = 'MS25_Jerez0707.mat';
            variable = 'MS25_Jerez0707';
        otherwise
            warning('Opción no reconocida: %s',
app.SetupDropDown.Value);
            return;
    end


    try
        datos = load(archivo, variable);  % ← Solo carga la variable
deseada
        app.MS25 = datos.(variable);      % ← Accede dinámicamente a la
variable cargada


        actualizarVista(app);
        SliderValueChanged(app, []);
    catch ME
        errordlg(['No se pudo cargar el archivo: ', archivo], 'Error');
        disp(ME.message);
     end
            end
        end

        % Component initialization
        methods (Access = private)

            % Create UIFigure and components
            function createComponents(app)

                % Create UIFigure and hide until all components are
created
                app.UIFigure = uifigure('Visible', 'off');
                app.UIFigure.Position = [100 100 640 480];
                app.UIFigure.Name = 'MATLAB App';

                % Create UIAxes
                app.UIAxes = uiaxes(app.UIFigure);
                title(app.UIAxes, 'Moto')
                xlabel(app.UIAxes, 'X')
                ylabel(app.UIAxes, 'Y')
                zlabel(app.UIAxes, 'Z')
```

```
            app.UIAxes.FontName = 'Arial';
            app.UIAxes.LineWidth = 1;
            app.UIAxes.Box = 'on';
            app.UIAxes.XGrid = 'on';
            app.UIAxes.YGrid = 'on';
            app.UIAxes.ColorOrder = [0 0.6471 0.7725;0.1647 0.2314
0.3608;0.9804    0.7255    0.1725;1    0.3412    0.2;0.4196    0.5569
0.1373;0.5019 0 0.5019;1 0.6275 0.4784];
            app.UIAxes.Position = [17 77 343 221];

            % Create UIAxes2
            app.UIAxes2 = uiaxes(app.UIFigure);
            title(app.UIAxes2, 'Circuito')
            xlabel(app.UIAxes2, 'X')
            ylabel(app.UIAxes2, 'Y')
            zlabel(app.UIAxes2, 'Z')
            app.UIAxes2.FontName = 'Arial';
            app.UIAxes2.LineWidth = 1;
            app.UIAxes2.Box = 'on';
            app.UIAxes2.XGrid = 'on';
            app.UIAxes2.YGrid = 'on';
            app.UIAxes2.ColorOrder = [0 0.6471 0.7725;0.1647 0.2314
0.3608;0.9804    0.7255    0.1725;1    0.3412    0.2;0.4196    0.5569
0.1373;0.5019 0 0.5019;1 0.6275 0.4784];
            app.UIAxes2.Position = [328 284 300 185];

            % Create SliderLabel
            app.SliderLabel = uilabel(app.UIFigure);
            app.SliderLabel.HorizontalAlignment = 'right';
            app.SliderLabel.Position = [109 37 36 22];
            app.SliderLabel.Text = 'Slider';

            % Create Slider
            app.Slider = uislider(app.UIFigure);
            app.Slider.Limits = [1 100];
            app.Slider.ValueChangedFcn   =   createCallbackFcn(app,
@SliderValueChanged, true);
            app.Slider.Position = [166 46 392 3];
            app.Slider.Value = 1;

            % Create LapSpinnerLabel
            app.LapSpinnerLabel = uilabel(app.UIFigure);
```

```
            app.LapSpinnerLabel.HorizontalAlignment = 'right';
            app.LapSpinnerLabel.Position = [129 330 25 22];
            app.LapSpinnerLabel.Text = 'Lap';

            % Create LapSpinner
            app.LapSpinner = uispinner(app.UIFigure);
            app.LapSpinner.Limits = [1 3];
            app.LapSpinner.ValueDisplayFormat = '%i';
            app.LapSpinner.ValueChangedFcn                          =
createCallbackFcn(app, @LapSpinnerValueChanged, true);
            app.LapSpinner.Position = [169 330 53 22];
            app.LapSpinner.Value = 1;

            % Create SetupDropDownLabel
            app.SetupDropDownLabel = uilabel(app.UIFigure);
            app.SetupDropDownLabel.HorizontalAlignment = 'right';
            app.SetupDropDownLabel.Position = [158 423 36 22];
            app.SetupDropDownLabel.Text = 'Setup';

            % Create SetupDropDown
            app.SetupDropDown = uidropdown(app.UIFigure);
            app.SetupDropDown.Items  =  {'Aceleracion',  'Carrera',
'Jerez07/07'};
            app.SetupDropDown.ValueChangedFcn                       =
createCallbackFcn(app, @SetupDropDownValueChanged, true);
            app.SetupDropDown.Position = [109 379 134 26];
            app.SetupDropDown.Value = 'Carrera';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = Pintamoto

            % Create UIFigure and components
            createComponents(app)
```

```
            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```