UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SISTEMA DE TRANSPORTE DE MATERIAL CON GRUPO DE ROBOTS VOLADORES MINIATURA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

MANUEL IGNACIO IGLESIAS CONTRERAS

PROFESOR GUÍA:
JUAN CRISTOBAL ZAGAL MONTEALEGRE

MIEMBROS DE LA COMISIÓN:
MARTÍN ADAMS
BRUNO GROSSI CÓRDOVA

SANTIAGO DE CHILE
2015

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: MANUEL IGNACIO IGLESIAS CONTRERAS
FECHA: MARZO 2015
PROF. GUÍA: Sr. JUAN CRISTOBAL ZAGAL

SISTEMA DE TRANSPORTE DE MATERIAL CON GRUPO DE ROBOTS VOLADORES MINIATURA

El objetivo general del presente trabajo es el de implementar un sistema de cuadricopteros miniatura controlados de forma autónoma, en un ambiente cooperativo, para el transporte de algún tipo de material de forma acumulativa. Se busca estudiar los posibles rangos de aplicaciones que puedan desarrollar cuadricopteros de tamaño reducido, que no sean factibles para drones de mayor tamaño, en cuanto a su desempeño, movilidad y escalabilidad.

El sistema consiste en cuadricopteros miniatura Crazyflie controlados utilizando teoría de control PID. Para realizar el feedback del lazo cerrado de control, se utilizara un sistema de captura de movimiento OptiTrack, un conjunto de cámaras para obtener la ubicación espacial de los cuadricopteros y todos los objetos pertinentes en la escena. El sistema de OptiTrack funciona junto a su software Motive:Tracker, el cual puede hacer transmisión de la información de la escena en tiempo real. Se escribe un programa en lenguaje de programación Python para recibir e interpretar toda la información recibida de la escena, implementar el control PID y crear una rutina de comportamiento para los cuadricopteros. Respecto al hardware, ha de configurarse el set de cámaras, ensamblar los cuadricopteros y diseñar el mecanismo de sujeción que utilizaran para para transportar carga.

Se logra implementar el sistema, desarrollando una iteración en el control de los cuadricopteros para adecuarse al propósito de transporte de material y trabajar acorde a las condiciones de su ambiente. Se consigue levantar 7gr de peso adicional, considerando que un cuadricoptero pesa 19gr. De la misma forma, el diseño del transporte de carga se ha iterado hasta lograr un funcionamiento adecuado, moviendo aproximadamente 2.7gr de arena por viaje.

AGRADECIMIENTOS

Quiero comenzar por agradecer profundamente a mi familia, en especial a mis padres Manuel Iglesias y Beatriz Contreras por todo el apoyo incondicional que me han dado durante mi formación académica. A su manera cada uno, siempre me impulsaron a no rendirme y continuar avanzando.

A mis amigos y compañeros de carrera. Agradezco a Javier Acuña, amigo con quien he compartido durante toda la carrera, por su amistad y los altibajos vividos en los estudios desde mechones. Luego también llegaron los amigos que conocí en eléctrica a los que también quiero mencionar y agradecer: Yerko Garrido, Boris Garrido, Felipe Zúñiga, Paulina Flores, Cristóbal Inostroza, Javier Sagredo, etc. Agradezco a todos ellos, a los que ya han salido y a los que aún les queda un poco, las experiencias compartidas.

Quiero agradecer a los Profesores de la comisión por promover la oportunidad de idear y realizar un Trabajo de Título de carácter más bien "práctico", el cual resultó ser un desafío especial ("no hay plan B"), donde abarqué diversos tópicos y etapas significativas de la carrera.

Por último, agradecer a Miguel Patiño, en el Laboratorio de Electrotecnologías por su ayuda al ensamblar los cuadricopteros, y a los compañeros de mecánica con los que compartí en el ya desmantelado ‘Laboratorio de Síntesis de Maquinas Inteligentes’ por el apoyo y las pequeñas sugerencias e ideas.

TABLA DE CONTENIDO

TABLA DE FIGURAS

1. INTRODUCCIÓN

Hoy en día los drones son aparatos voladores no tripulados que son vistos como una gran tendencia tecnológica en ascenso, siendo integrados en varios ámbitos de trabajo e investigación. El término "drone" no es reciente, si no que viene de hace casi un siglo atrás, como máquinas de uso militar, usadas para entrenamiento de cañones antiaéreos después de la primera guerra Mundial. Actualmente sus aplicaciones se han diversificado bastante, gracias a la continua reducción de costos y la aparición de nuevos avances tecnológicos, desde fines informativos, como mapeo de zonas geográficas para cartografía, patrullaje y rescate, evaluación de obras, exploración de lugares peligrosos, a simple recreación para el usuario civil.

Existen cuadricopteros de diversos diseños, tamaños y precios. Existen cuadricopteros miniatura que caben en la palma de una mano, muy novedosos pero con limitaciones claras frente a cuadricopteros de tamaño estándar, de gran dimensión. Aun así, un grupo de cuadricopteros miniatura podría resolver una tarea en un trabajo compartido, así como un grupo de abejas pueden recolectar polen y llevarlo al panal, que tras un procesamiento natural es usado para expandirlo y mantenerlo.

El objetivo general del proyecto presente es el de implementar un sistema de cuadricopteros miniatura controlados de forma autónoma, en un ambiente cooperativo, para el transporte de algún tipo de material de forma acumulativa. Se busca estudiar los posibles rangos de aplicaciones que puedan desarrollar cuadricopteros de tamaño reducido, que no sean factibles para drones de mayor tamaño, en cuanto a su desempeño, movilidad y escalabilidad.

Para lograr esto, los objetivos específicos planteados son el controlar un cuadricoptero mediante el uso del dispositivo OptiTrack, para luego extender esto y poder implementar un sistema con control automatizado que manipule un grupo de cuadricopteros. Así, se estudiará e implementará un comportamiento de trabajo de tipo cooperativo entre los cuadricopteros. Los cuadricopteros necesitaran algo con que interactuar con su ambiente, por lo

que se diseñará y construirá un equipo de sujeción de carga, implementando la habilidad de tomar y dejarla, sin intervenir los cuadricopteros mayormente, para probar su utilidad de forma simple. Finalmente, luego de realizar esto, se analizará el desempeño de la implementación en tiempo, control y escalabilidad.

El documento presente está estructurado en 5 capítulos. En el capítulo 2 se entregan los antecedentes generales, necesarios para comprender el funcionamiento de todo el sistema. Se describen los tópicos de hardware y los fundamentos teóricos. En el capítulo 3 se describe el proceso de la implementación del sistema en su completitud, cuadricopteros, sensores y su ambiente. En el capítulo 4 se presentan los resultados de las pruebas realizadas para evaluar los alcances de la implementación. Finalmente, en el capítulo 5 se presentan las conclusiones del trabajo realizado, con un breve análisis y proyecciones para su continuación en trabajos futuros.

2. CONTEXTUALIZACIÓN

En este capítulo se describen los conceptos generales básicos que dan marco al proyecto, colocándolo en un contexto. Tópicos referentes a hardware, software y fundamentos teóricos para la implementación del sistema.

2.1. Cuadricoptero

Un cuadricoptero, o *quadrotor*, es un helicóptero que se sostiene y propulsa con 4 rotores independientes dispuestos a distancias equivalentes, usualmente ubicados en las esquinas de una configuración cruz. Estos 4 rotores están posicionados de forma vertical, para la generación de propulsión.

Un cuadricoptero es considerado un vehículo aéreo no tripulado o UAV (*unmanned aerial vehicle*), categoría comúnmente conocida como drones. Los cuadricopteros utilizan 2 sets de hélices y rotores, girando uno en sentido horario y el otro en sentido anti horario, para así evitar un giro no deseado con respecto a su propio eje de orientación vertical.

El control del movimiento del vehículo se consigue variando la velocidad relativa de cada rotor para cambiar el empuje resultante, logrando 6 grados de libertad, con las inclinaciones Roll y Pitch, giro Yaw, y subida o bajada con modificaciones del Thrust.

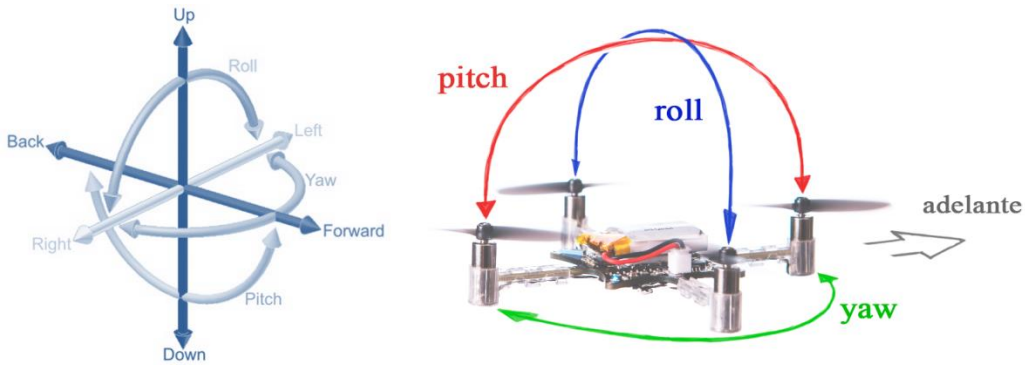


Figura 1: Grados de Libertad

2.2. Crazyflie

El Crazyflie Nano Quadcopter es un cuadricoptero miniatura, que pesa 19 gramos y mide 9 centímetros de motor a motor (motores opuestos).

Utiliza un microcontrolador STM32F103CB de 72MHz, con 128kb flash y 20kb de RAM. Alimentado con una batería Li-Po 170mAh, con un chip de radio de bajo consumo nRF24L01+. Incorporado con el sensor MPU-6050, que combina un giroscopio y un acelerómetro de 3 ejes en la misma placa [1] [2].

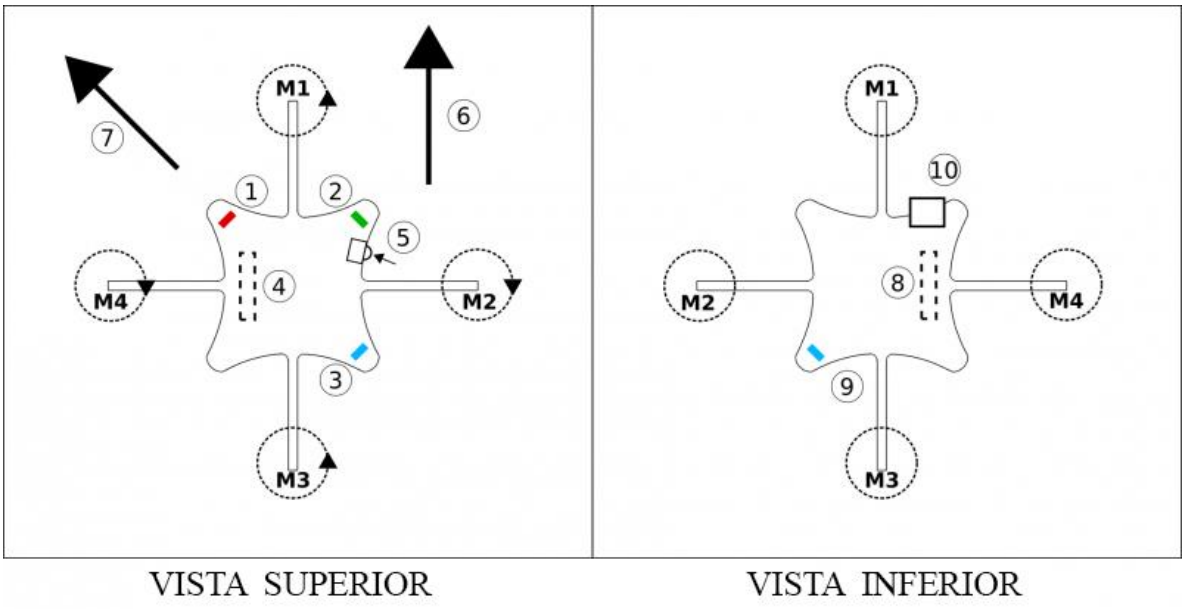


Figura 2: Esquema de Crazyflie

1) LED rojo - muestra estado del sistema y carga de batería.
2) LED verde - muestra estado de comunicación de radio.
3) LED azul - muestra estado de encendido.
4) Pines para posible expansión de funcionalidad.
5) Botón de encendido.
6) Dirección frontal al volar en una configuración normal.
7) Dirección frontal al volar en una configuración "X".
8) Pines para posible expansión de funcionalidad.
9) LED azul - muestra estado de encendido.
10) Micro USB para cargar la batería.

2.2.1. Manejo

Comercialmente, el cuadricoptero es manejable conectando un USB radio dongle y un control joystick a un equipo host, y corriendo el software cliente de Crazyflie. El software puede ser utilizado en un computador con sistema operativo Linux o Windows.

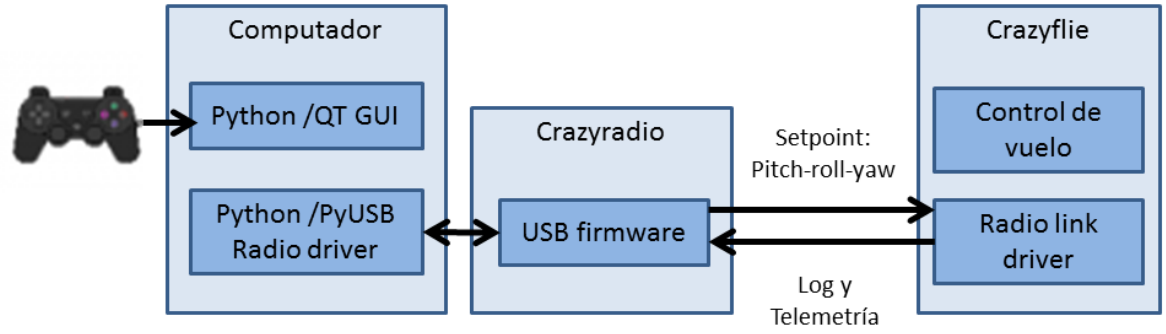


Figura 3: Diagrama de comunicación

El cuadricoptero se despacha como un kit a ensamblar, donde es necesario soldar los 4 motores a la PCB, así como fijar las monturas de los motores y la batería.

2.2.2. Vuelo

El Crazyflie tiene 4 controles diferentes: roll, pitch, yaw y thrust.

Cambiando roll o pitch, variando la velocidad entre motores opuestos, hará que el cuadricoptero se incline hacia sus lados, y así cambiando su dirección en la que se mueve. Inclinar el cuadricoptero hacia adelante provoca que se mueva hacia adelante, y de forma análoga para cualquier otra dirección. Cambiando yaw, variando la velocidad entre los motores de giro horario y los motores de giro anti horario, hará que el cuadricoptero gire en torno a su eje vertical. Finalmente, thrust es usado para controlar la altitud del cuadricoptero, al modificar la velocidad de los cuatro motores.

2.2.3. API de Crazyflie

API es una Interfaz de Programación de Aplicaciones (*Application Programming Interface)* lo que comprende a un conjunto de subrutinas, protocolos y herramientas para el desarrollo de software.

Una API puede tener la estructura de una librería que incluya las especificaciones para funciones, estructuras de datos, clases de objetos y variables pertinentes.

En este caso particular, Crazyflie cuenta con una librería hecha en Python que proporciona funciones de alto nivel para el uso del aparato [3].

2.2.3.1. Estructura de la librería

La librería está basada en callbacks para realizar un evento. Los callbacks son funciones que son llamadas como respuesta al ser entradas como parámetros para otras funciones. La librería no contiene ningún thread o lock que mantenga la aplicación funcionando por sí misma, por lo que la aplicación que haga uso de librería deberá encargarse de esto.

2.2.3.1.1. Uniform Resource Identifier (URI)

Todos los links de comunicación son definidos usando una URI construida de la siguiente manera:

Tipo de Interface://ID de Interface/Canal de Interface/Velocidad de Interface

Por ejemplo, la URI “radio://0/10/250K” define que la es utilizada la interfaz de radio para la comunicación, utilizado el USB dongle número 0, con el canal de radio 10 y a una velocidad de transmisión de 250Kbit/s.

2.2.3.1.2. Variables y logging

Logging es usado para el envío "automático" de valores de variables al cliente en intervalos especificados. La variable es un valor que es cambiada por el Crazyflie y no por el cliente, actualizándolas a altas tasas para ser leídas de forma periódica.

Las variables consisten del estado de la batería, como los estados de funcionamiento de los motores, y mediciones de los sensores incorporados.

2.2.3.1.3. Parámetros

Los parámetros son aquellos valores en el sistema que no pueden ser cambiados por el Crazyflie, sino que por el cliente. Los parámetros no son valores leídos periódicamente.

Los parámetros son considerados valores utilizados para la configuración del cuadricoptero antes de comenzar el manipulación, valores que restringen su comportamiento de vuelo, valores de identificación de sensores, de espacios de memoria y firmware.

2.2.3.2. Inicialización de link drivers

Antes de utilizar la librería se han de inicializar los drivers. Utilizando el siguiente código se buscan todos los drivers disponibles y se instancian.

```
    init_drivers(enable_debug_driver=False)
"""" Search for and initialize link drivers.
     If enable_debug_driver is True then the DebugDriver will also be used."""
```

2.2.3.3. Encontrar un Crazyflie

Se busca un cuadricoptero Crazyflie para conectarse a él. Las funciones de la librería escanean todas las interfaces de radio disponibles.

```
cflib.crtp.init_drivers()
   available = cflib.crtp.scan_interfaces()
   for i in available:
      print "Interface with URI [%s] found and name/comment [%s]" % (i[0], i[1])
```

Abrir y cerrar un link de comunicación con un Crazyflie es realizable de la siguiente manera:

```
crazyflie = Crazyflie()
  crazyflie.connected.add_callback(crazyflie_connected)
  crazyflie.open_link("radio://0/10/250K")
        ….
  crazyflie.close_link()
```

2.2.3.4. Envío de comandos de control

El envío de comandos de control al cuadricoptero no está implementado como el envío de “Parámetros”, debido a que debe hacerse a altas tasas. Para esto utilizan otra sección especializada de la API:

```
  send_setpoint(roll, pitch, yaw, thrust):
      """
      Send a new control set-point for roll/pitch/yaw/thust to the copter

      The arguments roll/pitch/yaw/trust is the new set-points that should
      be sent to the copter
      """
```

Se realiza de la siguiente forma:

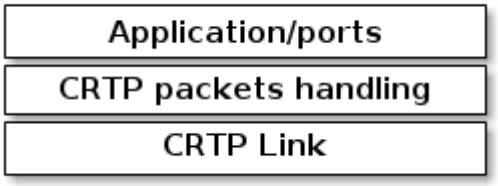
```
crazyflie.commander.send_setpoint(roll, pitch, yawrate, thrust)
```

El enviar un comando lo hace valido 500ms, a menos que se envié otro. Después de ese lapso, el firmware cortara la energía. Por esto, es necesario mantener el envío de comandos de forma constante para mantener encendido el cuadricoptero.

2.2.4. CRTP

CRTP o “Crazy Real Time Protocol” es el protocolo utilizado para realizar la comunicación con el Crazyflie [4]. El protocolo permite la comunicación de forma independiente con el subsistema del cuadricoptero y tener la capacidad de manejar la prioridad de los paquetes.

CRTP está implementado en 3 capas:



- CRTP Link se responsabiliza por la transferencia de los paquetes entre el cuadricoptero y el computador, manejando tamaños de paquetes y detección de errores.
- CRTP packets handling envía el paquete al subsistema correspondiente en el cuadricoptero y en el software de control del computador.
- Application/ports representa los subsistemas que envían y reciben mensajes. Dentro de los subsistemas principales, esta Commander.

2.2.4.1. Commander

El puerto para Commander es usado para el envío de Set-points de control para los reguladores de Roll, Pitch, Yaw y Thrust en el Crazyflie. Tan pronto como el link de comunicación ha sido establecido, estos paquetes pueden ser enviados, cuyos valores serán válidos hasta que el siguiente paquete sea recibido y el cuadricoptero aún se mantenga operativo.

```
                +-------+-------+-------+-------+
                | ROLL  | PITCH |  YAW  |THRUST |
                +-------+-------+-------+-------+
          Length     4       4       4       2     bytes
```

Name	Byte	Size	Type	Comment
ROLL	0-3	4	float	The pitch set-point
PITCH	4-7	4	float	The roll set-point
YAW	8-11	4	float	The yaw set-point
THRUST	12-13	2	uint16_t	The thrust set-point

2.3. Crazyradio

Crazyradio es un 2.4GHz radio USB dongle, diseñado para ser utilizado en conjunto con el cuadricoptero Crazyflie, estableciendo la comunicación entre cuadricoptero y el equipo host, enchufado en un puerto USB del computador controlador.

Utiliza una antena ajustable de conexión coaxial RP-SMA, para rango de frecuencia de 2.4GHz, 2db de ganancia e impedancia de entrada de 50 ohms.

Permite el uso de 125 canales de radio, a velocidades de transmisión de datos de 2Mbps, 1Mbps y 250Kps. Envía y recibe paquetes de datos de hasta 32 bytes, manejando automáticamente las direcciones y los paquetes ACK. El dongle permite un alcance máximo de 80 metros sin pérdida de señal.

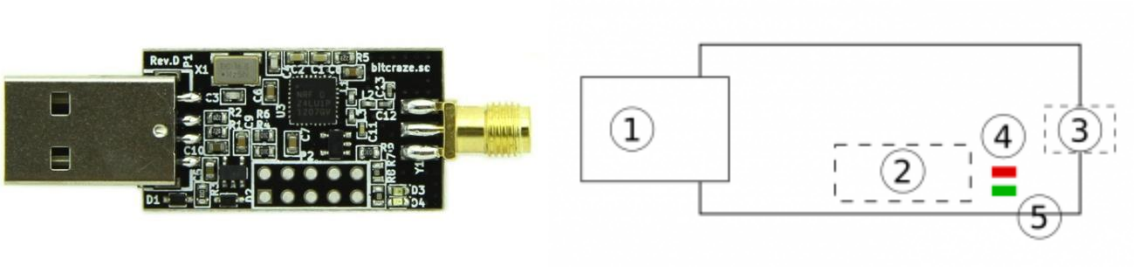


Figura 4: Crazyradio y esquemático simple

1) USB – Conexión a puerto de computador con estándar USB2.0.
2) Interfaz PPM para posible extensión de programación.
3) Conexión con antena.
4) LED rojo - parpadea cuando es posible enviar datos al cuadricoptero.
5) LED verde - parpadea cuando se han enviado datos al cuadricoptero.

2.4. OptiTrack

OptiTrack es un sistema de captura de movimiento y seguimiento de objetos en tiempo real. Está compuesto por un equipo 6DoF V120:Trio de 3 cámaras, y el software de captura Motive:Tracker.

V120:Trio consistente en un equipo de cuerpo de aluminio con 3 cámaras alineadas, conjunto a 26 ampolletas LEDs cada una. El equipo captura datos con una resolución de 640x480 pixeles, a una tasa de 120FPS y una latencia máxima de 8.3ms. Utiliza interfaz de conexión USB2.0

Figura 5: OptiTrack V120:Trio

Este sistema permite el rastreo de objetos en 6 grados de libertad con precisión de orden milimétrica (con variaciones aproximadas de $\pm 0.1mm$), brindando soporte para ambientes de trabajo en tiempo real u offline.

La versión utilizada del sistema permite la captura de los siguientes tipos de información: marcadores no agrupados y Cuerpos Rígidos. Un Cuerpo Rígido es un cuerpo solido indeformable ideal conformado por un conjunto mínimo de 3 marcadores seleccionados.

Para utilizar las funciones de tiempo real del sistema, el software Motive:Tracker provee un protocolo para realizar Streaming Data del tipo Direct UDP. El acceso a los datos capturados en tiempo real mediante este protocolo permite el streaming de todos los tipos de información previamente mencionados y el uso de lenguajes de programación C/C++ o Python para la creación del cliente UDP.

2.5. Data Streaming

El termino Data Streaming corresponde a la transmisión de información representada por paquetes de datos, una secuencia de señales codificada digitalmente.

2.5.1. User Datagram Protocol (UDP)

Protocolo para el envío de datagramas, donde el datagrama incorpora la información de direccionamiento suficiente en su cabecera, para realizar un envío a través de la red sin establecimiento previo de una conexión.

El protocolo utiliza la habilitación de puertos para permitir la comunicación entre aplicaciones. El campo de puerto tiene una longitud de 16 bits, por lo que el rango de valores validos va de 0 a 65.535.

El protocolo no presenta confirmación de entrega o recepción ni control de flujo de datos. Esto permite ser usado cuando la tasa de intercambio de paquetes demandada es alta. Ampliamente ocupado en tareas de control y en la transmisión de audio y visto a través de una red.

No introduce retardos para establecer una conexión, no necesita mantener el estado o realizar seguimientos de los parámetros de una. Como trabaja sin conexión, y no utiliza sincronización entre origen y destino, admite a un paquete UDP utilizar como dirección IP de destino la dirección de broadcast o de multicast de IP, permitiendo el envío de paquetes a varios destinos.

2.5.2. Direct UDP

Una conexión Direct UDP, refiere a una conexión del tipo UDP directa de un punto a otro, la cual no requiere pasar a través de terceros.

2.5.3. Multicast

Método de transmisión de datagramas IP, donde el direccionamiento multicast está asociado sólo con un grupo de receptores interesados, a diferencia del concepto de Broadcast. Según la publicación RFC 3171 [12], las direcciones multicast habilitadas para esta forma de envío de datos son desde la 224.0.0.0 a la 239.255.255.255.

2.6. Control PID

Un controlador PID calcula un valor error como la diferencia entre una variable medida de proceso y un set-point deseado. El controlador busca minimizar el error ajustando el proceso a través de una variable manipulable.

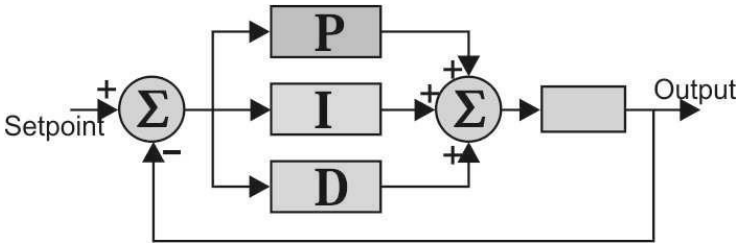


Figura 6: Esquema simple de control PID

El algoritmo del controlador hace uso de tres parámetros constantes, para el cálculo de tres términos: Proporcional, integral y derivativo. La suma ponderada de los tres términos se utiliza para ajustar el proceso y obtener el output deseado del sistema.

2.6.1. Término P: Proporcional

El término proporcional cambia el ouput con respecto al error. Cambiando el parámetro de ganancia proporcional (Kp), el control se vuelve más o menos reactivo a los cambios.

$$P = Kp * e$$

A más alto el parámetro Kp, el sistema intentara alcanzar el set-point más rápido, pero también una ganacia proporcional alta hace que al sistema inestable y que oscile entorno al set-point. Por otro lado, si se mantiene el Kp bajo, entonces el sistema se estabilizara con un error constante bajo (o sobre) el set-point, porque el output no será suficiente para que el sistema alcance su posición final esperada.

2.6.2. Término I: Integral

Un sistema no puede solo cambiar su output acorde a su error actual, pero también debe vigilar y cambiar su output acorde a los errores pasados. El término integral es la suma de los errores en el tiempo. Esto significa que si el error es grande, la integral ira incrementando conforme pase el tiempo y el ouput cambiará rápidamente para eliminar dicho error.

A modo de ejemplo, tomando un sistema proporcional con un Kp bajo, el sistema responderá más rápido ahora en su comienzo, dado que el error se ira sumando en una gran integral. Entonces, cuando el sistema se estabilice por debajo del set-point, la integral hará efecto. Este pequeño error se sumara en el tiempo para incrementar la integral, y finalmente esto cambiara el output del sistema. Al final, el sistema llegara a estabilizarse en el set-point.

El término integral es multiplicado por el parámetro de ganancia integral para incrementar o reducir el efecto, tomando la siguiente forma:

$$I = Ki * \int e$$

Una ganancia integral (Ki) alta causa una respuesta del sistema más rápida, pero también causa un overshoot. A diferencia de un sistema netamente proporcional, a pesar del overshoot, el sistema no será llevado a un estado inestable. Tras algunas oscilaciones, el sistema debe estabilizarse en el set-point. Con una ganancia integral baja, el sistema responderá más lento, pero se estabilizara más rápido con menos oscilaciones. Una ganancia integral óptima causara que el sistema responda lento, pero se estabilizara en el set-point casi sin presencia de oscilaciones.

2.6.3. Término D: Derivativo

El tercer término es el derivativo, el cual calcula la tasa de cambio del error y altera el output de forma acorde. Si el error cambia lentamente, el parámetro Kd ha de ser incrementado para que el sistema PID responda más rápido. Por otro lado, si el error cambia rápidamente, el parámetro ha de reducirse para hacer al sistema más estable y evitar oscilaciones. La ecuación que describe el término derivativo es:

$$D = Kd * (e_n - e_{n-1})$$

El ouput de controlador PID completo es dado por la siguiente forma:

$$Pout = P + I + D = Kp * e + Ki * \int e + Kd * (e_n - e_{n-1})$$

2.6.4. PID de Velocidad

El PID de Velocidad es un algoritmo utilizado para realizar control de lazo cerrado de la velocidad en un eje de movimiento.

El termino Proporcional corresponde al Error de velocidad, es decir a la diferencia entre la velocidad actual y la especificada por el Set-Point. El término Integral es proporcional al Error Acumulado de velocidad, equivalente a la variación de la posición. Y el término Derivativo, proporcional a la variación de aceleración.

2.6.5. PID de Posición

El PID de Posición es un algoritmo utilizado para realizar control de lazo cerrado sobre la posición en un eje de movimiento. Sigue de forma eficiente el posicionamiento, pero puede oscilar ante un objetivo de movimiento abrupto.

El término Proporcional corresponde a la diferencia entre la posición actual y la estipulada por el Set-Point, el Error de posición. El termino Derivativo, es proporcional a la variación de la velocidad. Finalmente, el termino Integral, es proporcional al Error Acumulado de la posición.

2.7. Stigmergy

Se entiende el concepto de Stigmergy como un mecanismo de coordinación entre agentes, donde los cambios provocados en el ambiente por alguna acción, estimula el desarrollo de una siguiente acción o acciones futuras varias, por el mismo individuo u otro [17].

Otra forma de verlo, es como un modo de comunicación en la que los individuos se comunican entre sí por medio de la modificación de su ambiente local, una clase de colaboración, ante la falta o la inefectividad de realizar una negociación previa [15].

El término fue utilizado por vez primera en la década de 1950 por Pierre-Paul Grassé en una investigación sobre comportamiento en termitas [17].

Existe el precedente de Robots inspirados en Termitas, que implementan el concepto de Stigmergy para poder trabajar [16].

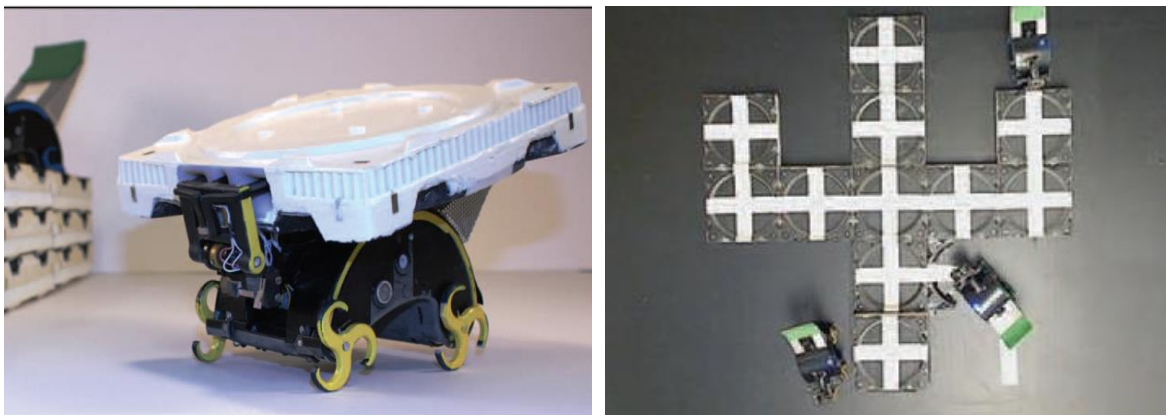


Figura 7: Robots termitas [16]

2.7.1. Comportamiento esperado del sistema

Se espera poder visualizar de cierta forma una colaboración del tipo Stigmergy en el sistema a implementar, entendiendo las distintas condiciones y limitaciones existentes. A continuación, un esquema del funcionamiento de la comunicación que tendrá el sistema.

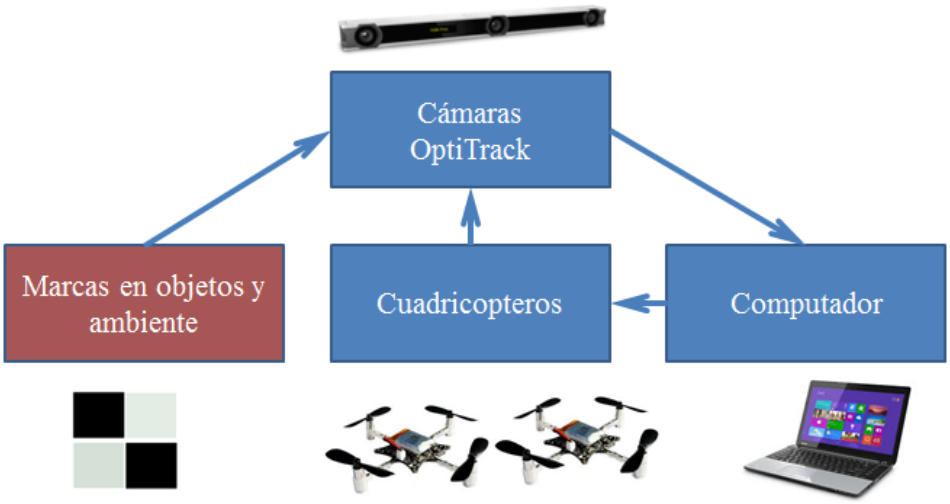


Figura 8: Diagrama de funcionamiento del sistema

Los cuadricopteros se desplazaran a buscar cargas a trasportar y depositar. Para ello los cuadricopteros necesitan reaccionar a cambios en su ambiente. En particular, cada cuadricoptero reacciona al cambio de posición en tiempo real del lugar donde encontrar una carga a transportar como del lugar a liberar su carga.

Por ejemplo, si el lugar de depósito del material es movido a cualquier otro punto por la acción de un tercero, el viaje de turno es modificado para llegar efectivamente donde se necesita.

De igual manera, siempre está presente una reacción a la variación de la posición del material a transportar. Cada vez que un cuadricoptero elige y transporta una carga, la cantidad de cargas libres a transportar disminuye, con lo que el viaje para recolectar una nueva carga se puede ver modificado al que se vio planeado inicialmente para el siguiente viaje. Cada vez que se toma una carga y se realiza un viaje, los agentes reaccionan a ello para elegir la siguiente carga y realizar los viajes a continuación. Es decir, los viajes y trayectorias son establecidos en función del ambiente actual de trabajo.

3. METODOLOGÍA

En este capítulo se describirán las actividades del plan de trabajo que permitieron desarrollar el proyecto. Cabe aquí mencionar:

1. Controlar un cuadricoptero mediante el uso del OptiTrack. Esto consiste de:

    - Ensamblar cada cuadricoptero y comprender el funcionamiento del sistema de radiocontrol que utilizan de fábrica.

    - Implementar un software de control en base a la API del Crazyflie y la utilización del sistema de cámaras OptiTrack.

2. Implementar sistema de control autónomo para un grupo de cuadricopteros, realizando distintas pruebas y ajustes a las variables de control. El objetivo consiste en manipular y comandar más de un cuadricoptero a la vez.

3. Estudiar e implementar una rutina de comportamiento de trabajo de tipo cooperativo. Esto comprende la creación de un sistema de reglas de decisión para el control principal.

4. Diseñar, construir y acoplar mecanismo de sujeción de objetos a cuadricopteros, implementando la habilidad de tomar y dejar carga.

3.1. Hardware

3.1.1.Ensamble de cuadricoptero

Antes de ensamblar el cuadricoptero, se realizó un rápido chequeo a la placa de control y la conexión con el Crazyradio [5]:

- Se alimenta la placa del Crazyflie con un cable micro-USB. Al encenderse, esto se verifica con el LED verde que debería parpadear 5 veces. Luego, el LED verde debería quedarse completamente encendido y el LED rojo debería parpadear.

- Se enchufa el Crazyradio a un computador y se instala. Cuando el Crazyradio es identificado, los LEDs verde y rojo se apagan.

Las piezas en el kit necesarias para el ensamblado eran las siguientes:
PCB cuerpo, 4 motores, 4 monturas de motor, 2 hélices de giro horario, 2 hélices de giro anti horario, una batería LiPo.

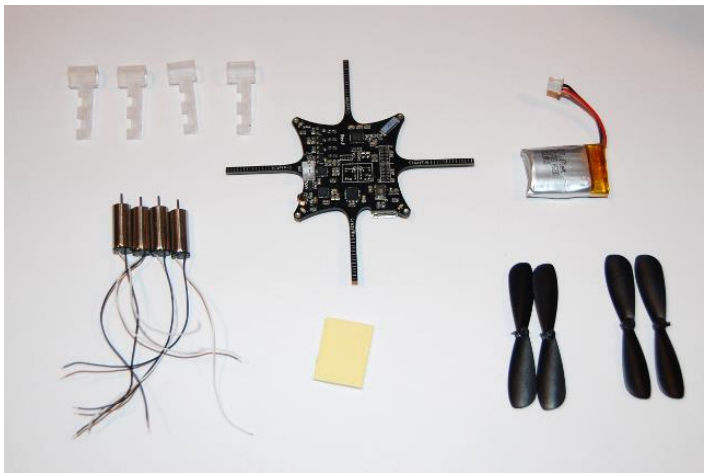


Figura 9: Kit desarmado de Crazyflie

Se enrollaron juntos los cables de los motores, para así facilitar su manipulación (y reducir el ruido electromagnético).

Se pasaron los cables enrollados del motor por una montura de motor, y se terminó de insertar el motor en la montura. Se presionó la montura en uno de los brazos de la PCB, siempre tomando precaución de no dañar los cables. Esto se realizó con las 4 monturas.

Figura 10: Posicionamiento de motores en PCB

Los extremos de los cables se insertaron en el orificio del pad correspondiente. Este proceso se realizó con los 4 pares de cables, siguiendo el posicionamiento de cada motor y alternando el sentido de los cables según se especifica en la Figura 11. Esto permite el funcionamiento correcto del giro de cada motor.

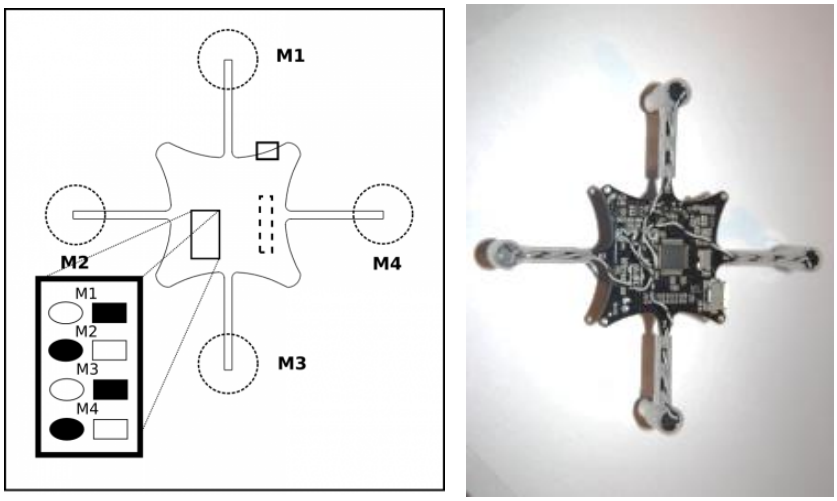


Figura 11: Esquema de soldado de motores por lado inferior y su resultado

Se soldaron los 4 pares de cables en sus pads correspondientes. Debido a la precisión necesaria, se utilizó una estación de soldadura digital, con una temperatura controlada de 250°C, y cabezal ajustable.

Las hélices se colocaron siguiendo la Figura 12. Para ello, existen 2 tipos diferentes de hélices, donde los de giro horario tienen rotulados una letra “A”. Motores 2 y 4 utilizan hélices de giro horario, y los motores 1 y 3 utilizan hélices de giro anti horario.

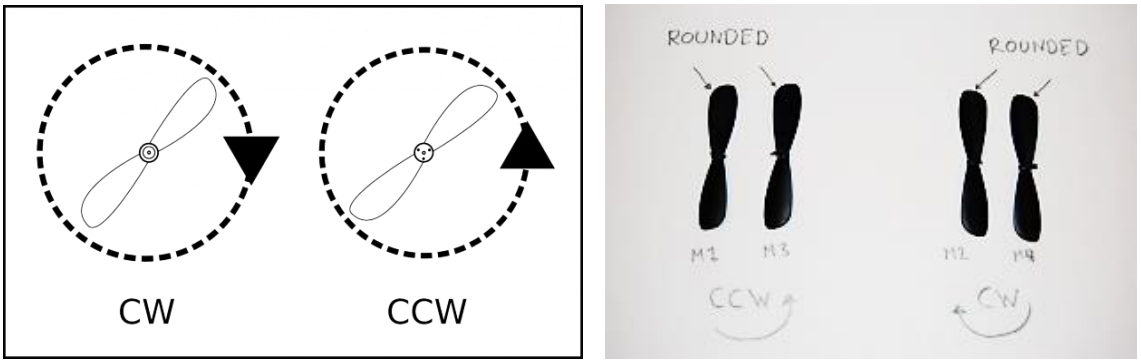


Figura 12: Sentido de las hélices

Se afirmó la batería LiPo en la PCB con una cinta engomada, proporcionada con el kit de ensamblado. Se cuidó de centrar la batería en la PCB con el objetivo de centralizar el peso total del cuerpo del cuadricoptero.

Antes de alimentar el cuadricoptero, se inspeccionó visualmente los puntos soldados, asegurándose de que el cobre de los cables no haga contacto con las resistencias cercanas. Se utiliza un multímetro y se chequea una resistencia de 2 ohms entre cada par de pads de los motores para verificar su correcto ensamblado.

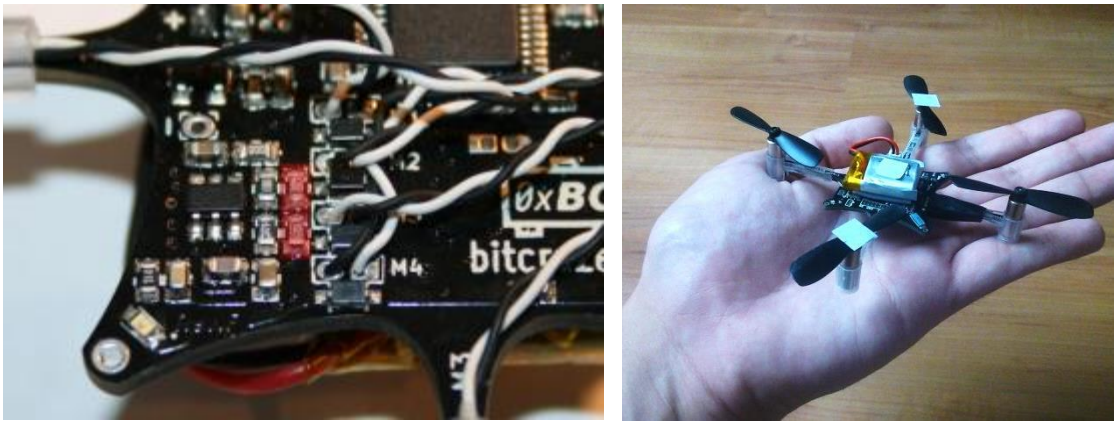


Figura 13: Verificación del armado

3.1.2.Operación del OptiTrack

El equipo 6DoF V120:Trio de 3 cámaras estaba fijado en el techo del Laboratorio de Síntesis de Maquinas Inteligentes, 5° piso de Mecánica, aproximadamente a una distancia de 1,83m sobre la superficie de la mesa central. Se capturó la escena del sistema a través de las 3 cámaras conjuntas, lo que se ve reflejado en tiempo real en el software Motive:Tracker.

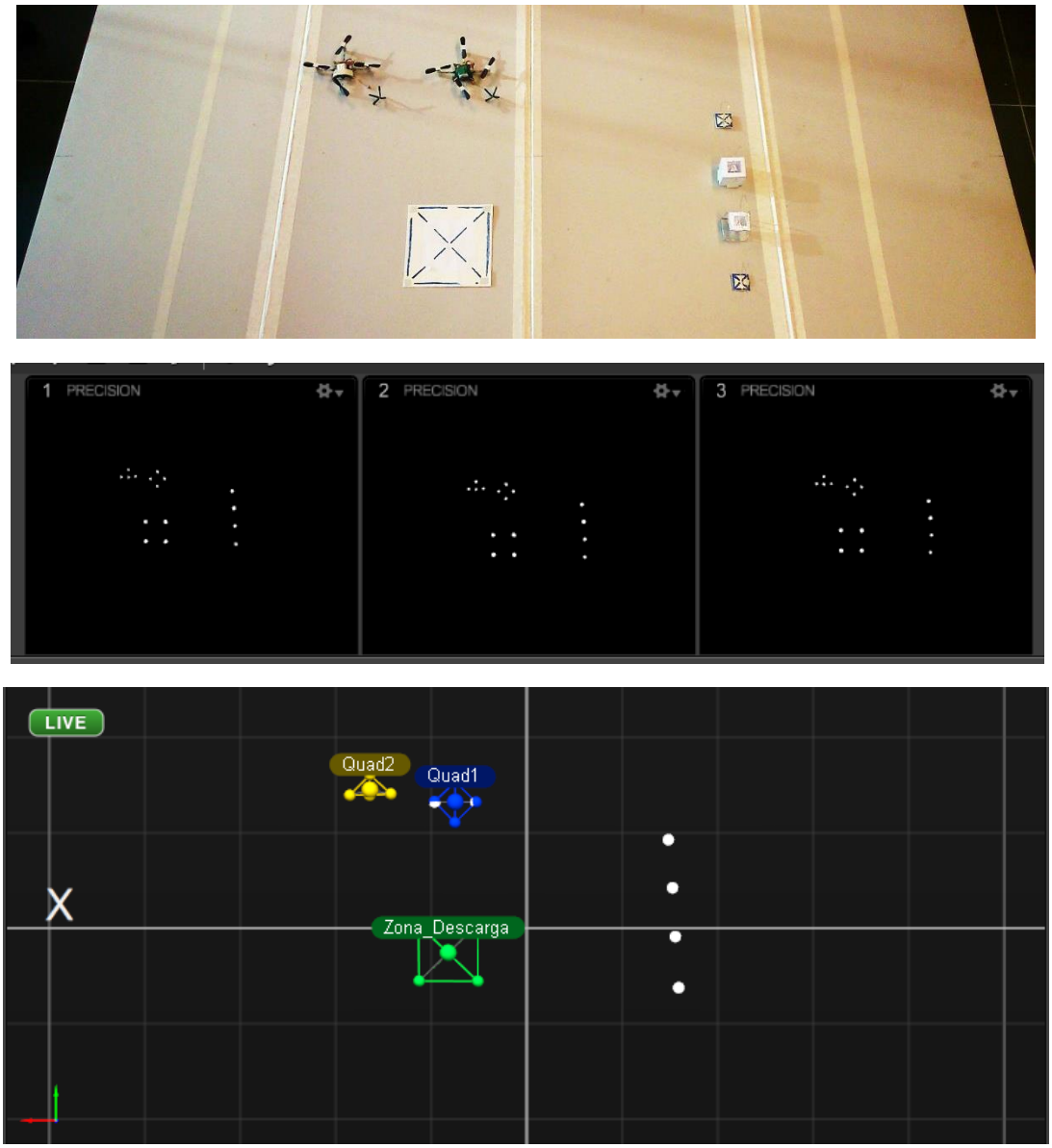


Figura 14: Vista de las cámaras Optitrack

Los marcadores para la escena del sistema son recortes cuadrados de una cinta autoadhesiva de material reflectante de infrarrojos. Esto permitió poner los marcadores necesarios sobre los elementos que interactúan en el sistema, sin adicionar volumen o peso extra., siendo esto un aspecto fundamental en este proyecto.

Los cuadricopteros, las cargas y las zonas delimitadas de interacción fueron marcadas para ser reconocibles por el OptiTrack.

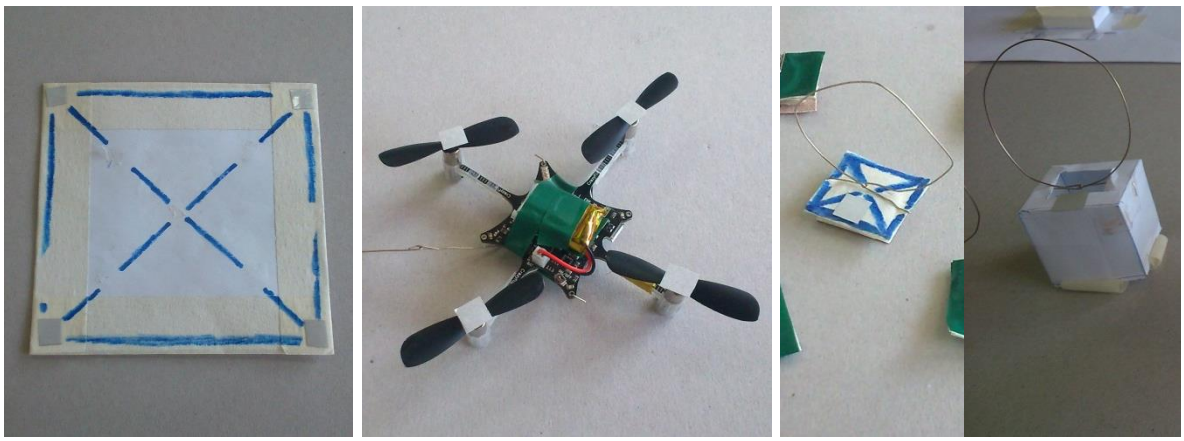


Figura 15: Marcadores reflejantes

La cantidad de marcadores en la escena y la asociación de estos a una cantidad de Cuerpos Rígidos, afectaban el comportamiento del programa Motive:Tracker. El tamaño de información capturado por segundo, la latencia en su desempeño, y el tiempo que toma su procesamiento aumentan acorde. Para una escena promedio de 3 Cuerpos Rígidos de 4 marcadores asociados cada uno y 5 marcadores indefinidos, la información promedio capturada por segundo era de 750KB, generando una latencia promedio de 1.1ms. El tiempo de procesamiento adicional de la asociación de marcadores bordeaba los 0.4ms.

Cabe destacar que la captura correcta de la escena era dada por una configuración de las cámaras acorde a las condiciones lumínicas del laboratorio.

- El Frame Rate (FPS) se dejó fijado en 120FPS, su valor más alto para esta clase de cámaras. Se descartaron las tasas opcionales de 60FPS o 30FPS por limitar el control óptimo del cuadricoptero.
- La iluminación LED (LED) se fijó en su máximo, para la correcta captura de los marcadores en la escena.
- El umbral o Threshold (THR) define el brillo mínimo para que un pixel sea visto por una cámara, mientras que los pixeles con menor brillo que el umbral serian ignorados. Dada la naturaleza de los marcadores reflexivos y el contraste con la superficie de trabajo de la escena, se optó por un valor superior, para descartar reflejos y fuentes de luces externas. Un valor bajo permitiría ver marcadores más pequeños y opacos, pero al coste de confundir fuentes de luces no controlables.
- El tiempo de exposición (EXP) para cada frame. Este parámetro se regula según la luminosidad del laboratorio a lo largo del día. A mayor luz, menor el valor necesario para capturar una imagen nítida. Pero si se elige un valor muy alto, pueden producirse la aparición de falsos marcadores.

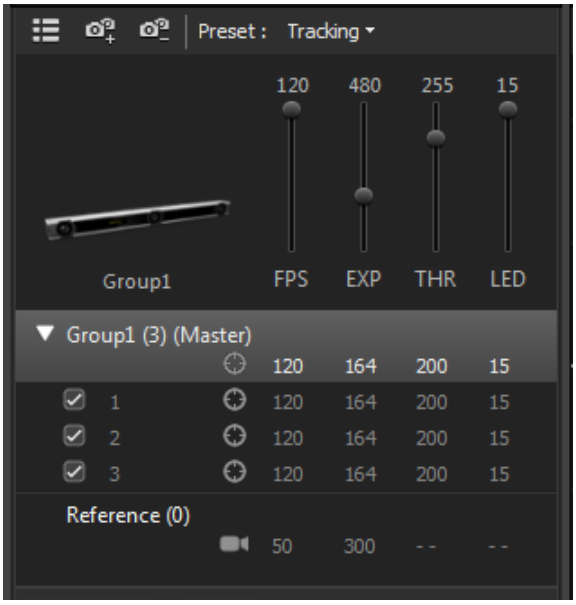


Figura 16: Panel de configuración de cámaras [9]

El programa fue configurado para realizar un streaming en tiempo real. La información a ser enviada es llamada "Frame Data" [10], correspondiente a todos los datos capturados en la escena dentro del campo visual de las cámaras, frame a frame.

En la ventana de opciones de captura de datos, en Streaming Propierties, se activa la opción Broadcast Frame Data, donde el programa empieza a ejecutar su engine propietario de streaming. Se configuran los siguientes campos para el método y protocolo.

Se habilitan el stream de Marcadores y Cuerpos Rígidos. Esto permite hacer streaming de la posición espacial relativa de los marcadores individuales, y el nombre, posición y orientación de los marcadores asociados a cuerpos rígidos.

Se seleccionan las opciones del tipo de broadcast, y el puerto que será usado para el streaming de la información desde el servidor al cliente, conjunto con la dirección del multicast broadcast.

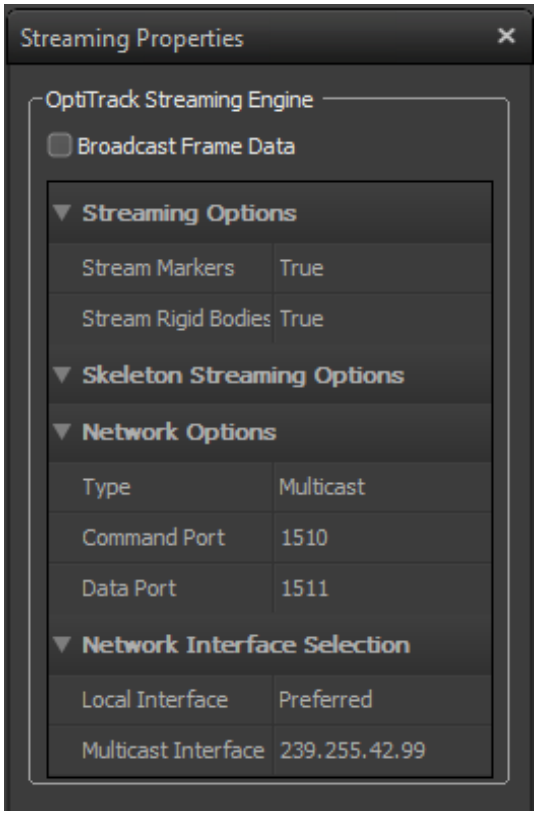


Figura 17: Panel de propiedades de Streaming [10]

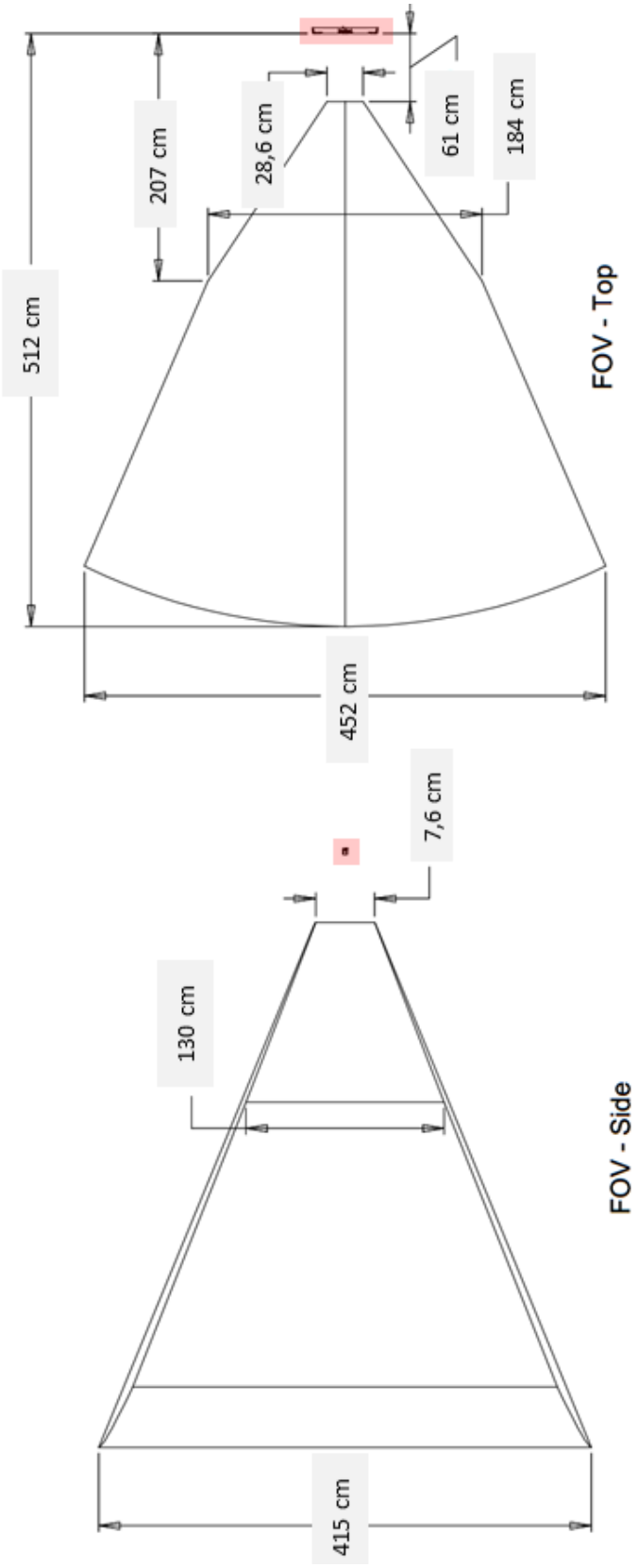


Figura 18: Point of View del Optitrack

3.2. Software

El programa para controlar el sistema se implementó en lenguaje de programación Python, dada la convergencia que existe entre las librerías de desarrollo tanto del Crazyflie como del OptiTrack [3] [11]. Otra característica a favor de este lenguaje, era la posibilidad de una rápida implementación de software con características de prototipo, para su fácil y rápido desarrollo, modificación y expansión.

La implementación del programa para controlar el sistema sigue un funcionamiento general dado por la siguiente estructura. El programa está dividido en Threads, los cuales permiten el desarrollo de sus tareas auto contenidas en forma simultánea, lo cual es de vital importancia para el funcionamiento en tiempo real del sistema.

El programa está compuesto por:

- Thread 0: Captura de datos desde el OptiTrack e interpretación
- Thread 1: Conexión, comportamiento y control de cuadricoptero 1
- Thread 2: Conexión, comportamiento y control de cuadricoptero 2

3.2.1.Cliente Streaming

Se realizó la recepción del streaming mediante la implementación de un cliente para el broadcast que realiza el programa Motive:Tracker con los datos capturados desde el Optitrack. Para ello se habilitaron y especificaron la dirección y puertos a los que hay que escuchar, conjunto con el formato de paquetes esperados.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('', 1511))

mreq = struct.pack("=4sl", socket.inet_aton("239.255.42.99"), socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
sock.settimeout(1)  #borrable si se requiere esperar por data
```

El cliente recibía el data streaming que realiza el programa Motive:Tracker, obteniendo el Frame Data del sistema. La información del Frame Data recibida estaba codificada. La interpretación del string fue implementada en el método unPackData().

El método unPackData() utiliza la función de Python unpack(),

```
struct.unpack (fmt, buffer[, offset=0])
```

la cual recibe como parámetros el *packed string* y el formato que debería tener originalmente. El string debe contener exactamente la cantidad de información requerida por el formato, donde se puede dictar la un substring que se quiere obtener en particular [13].

Por ejemplo, para obtener "frameNumber" y "nMarkerSets", 2 variables que vienen contiguas en el string, se indica a la función que el formato de ambas variables es *integer* (i), y que se encuentran vienen en la posición indicada por "offset" hasta "offset+8".

```
frameNumber,nMarkerSets = unpack(byteorder+'ii',PacketIn[offset:offset+8])
```

Para utilizar correctamente esta función se necesitó saber a priori la compleja estructura del *Frame Data* recibido del programa Motive:Tracker, por lo que se recurrió a la API pertinente de OptiTrack [7].

3.2.2.Conexión a Crazyflie

Se utilizó un comando de escaneo de interfaces del protocolo CRTP, el cual busca Crazyflies encendidos disponibles para conectar. Cada Crazyflie detectado respondía al emisor con un mensaje detallando sus datos de identificación y conexión, los cuales son únicos por cada aparato.

Si la información recibida coincidía con la información que se tenía registrada de un cuadricoptero del sistema, se iniciaba un proceso de conexión con él.

Un objeto Crazyflie era inicializado con los datos de identificación del cuadricoptero, llamando por primera vez a las funciones de verificación de estado. Los 4 estados de conexión que puede presentar un cuadricoptero Crazyflie son: conectado, desconectado, conexión fallida, o conexión perdida. El estado inicial por default del aparato era "desconectado". Esta función permanecía funcionando en segundo plano, verificando en tiempo real el estado de la conexión.

Si no se lograba establecer un enlace con el cuadricoptero, se establecía un estado de "conexión fallida". Si se lograba un intercambio de llamadas exitoso, cambiaba el estado del cuadricoptero a "conectado", y se realizaba un callback a las funciones de control del aparato.

Al perder conexión con el cuadricoptero, en el caso de una emergencia, se generaba un cambio del estado de conexión a los estados "conexión perdida", y se informaba al resto del programa para detener la rutina de control.

3.2.3.Control de Crazyflie

El programa corría una rutina de “inicialización” y luego, un loop infinito “ciclo principal”, donde se calculaban los términos PID y así controlar el ouput. El diagrama de flujo correspondiente:

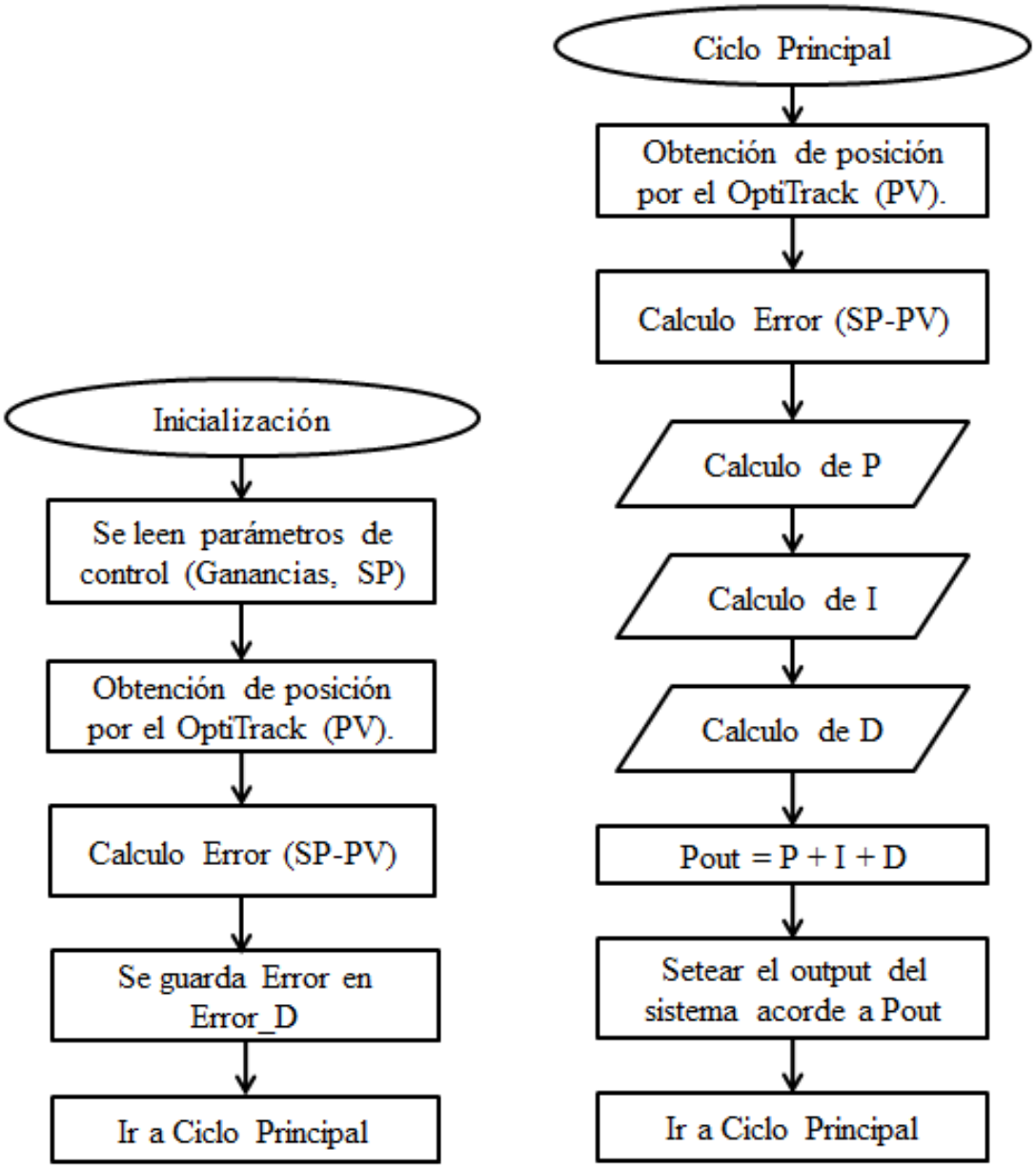


Figura 19: Diagramas de flujo del control en el programa

Existen 3 subrutinas, para los calculos de los terminos del control mismo.

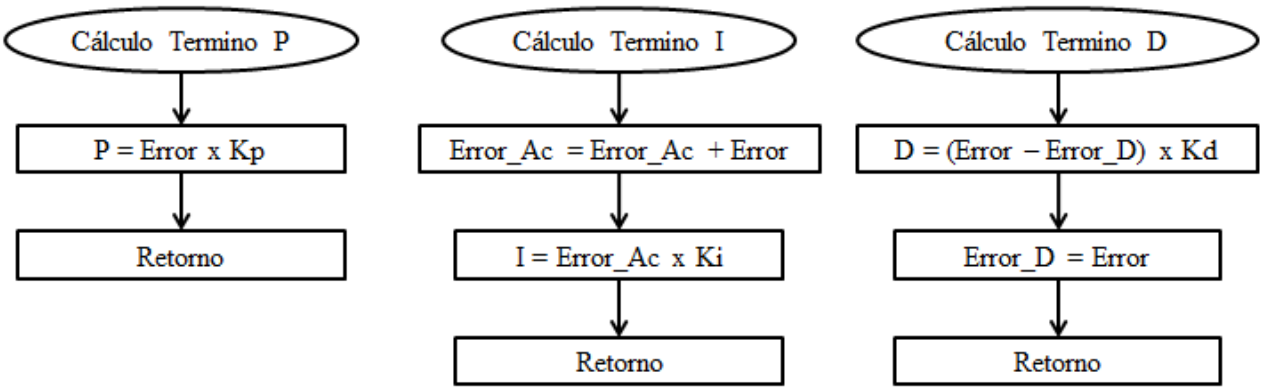


Figura 20: Diagramas de flujo de subrutinas del control

Para calcular los valores adecuados de roll y pitch a enviar, valores del control horizontal del cuadricoptero, se implementó 2 controles PID de Velocidad. Para ello, se siguió la siguiente implementación del cálculo.

$$P = Kp * (Velocidad_i - Velocidad_{SP})$$

$$I = Ki * (Posición_i - Posición_{SP})$$

$$D = Kd * (Aceleración_i - Aceleración_{SP})$$

Se ocupa el concepto de que el Set-Point utilizado para este control es de la siguiente naturaleza: Posición espacial (x,y,z) cualquiera dentro de los límites de la escena capturable por el OptiTrack del sistema, a Velocidad y Aceleración nulas. Las ecuaciones quedan representadas de la siguiente forma:

$$P = \text{Kp} * (Velocidad_i - 0) = Kp * (Posición_i - Posición_{i-1}) * cte1$$

$$I = Ki * (Posición_i - Posición_{SP})$$

$$D = \text{Kd} * (Aceleración_i - 0) = Kd * (Velocidad_i - Velocidad_{i-1}) * cte2$$

$$= Kd * (Posición_i - 2 * Posición_{i-1} + Posición_{i-2}) * cte3$$

Análogamente, para calcular el valor adecuado para el thrust a enviar, valor referente al control vertical del cuadricoptero, se implementó un control PID de Posición. Para ello, se siguió la siguiente implementación del cálculo.

$$P = Kp * (Posición_i - Posición_{SP})$$

$$I = Ki * \text{Error Acumulado de Posición}$$

$$D = Kd * (Velocidad_i - Velocidad_{SP})$$

Dado el Set-Point de la siguiente naturaleza: Posición espacial (x,y,z), a Velocidad y Aceleración nulas. Las ecuaciones quedan representadas de la siguiente forma:

$$P = Kp * (Posición_i - Posición_{SP})$$

$$I = Ki * \text{Error Acumulado de Posición}$$

$$D = \text{Kd} * (Velocidad_i - 0) = Kd * (Posición_i - Posición_{i-1}) * cte1$$

La elección de un PID de Posición fue realizada por la necesidad de lograr que el cuadricoptero pudiera hacer "hover", es decir estabilizarse perfectamente entorno a una posición vertical fija, para realizar un movimiento libre de forma horizontal.

Tan pronto como la comunicación con el cuadricoptero hubiera sido establecida, estos paquetes podían ser enviados y sus valores eran válidos hasta que el siguiente paquete fuera recibido.

La función de control implementada siguió una estructura donde por cada ciclo de cómputo, se conservaban los parámetros calculados u obtenidos para ser usados en el siguiente ciclo de la función, permitiendo realizar aproximaciones finitas para el cálculo de error acumulado para los términos Derivativos e Integrales.

La función setpoint del puerto commander, permite enviar al cuadricoptero los set-points de control para los reguladores de roll, pitch, yaw y thrust.

3.2.4.Rutina de Seguimiento

Por cada ciclo de cómputo, dentro del Thread de manejo del cuadricoptero, era posible llamar a la función implementada llamada follow() antes de realizar la acción de control. Ésta función actualizaba a voluntad el Set-Point de la posición del cuadricoptero para que se modificaran de forma acorde los valores de roll, pitch y thrust, y lograr el desplazamiento deseado del aparato.

La función follow() utilizaba las posiciones espaciales de los marcadores del cuadricoptero y un "puntero", los cuales eran capturados por el Optitrack.

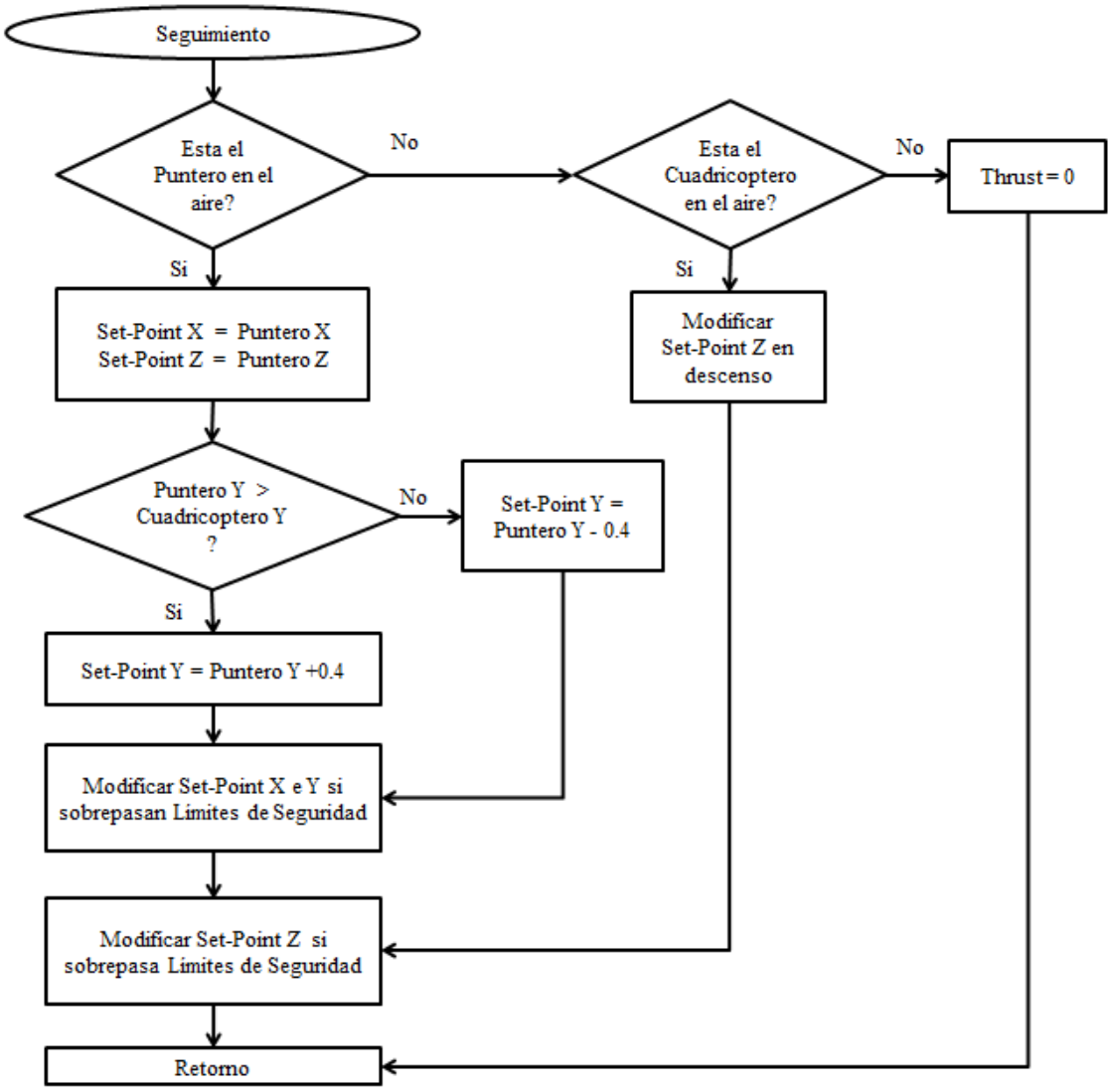


Figura 21: Diagrama de flujo de rutina de seguimiento

Bajo el control de esta rutina, el cuadricoptero era capaz de seguir al objeto "puntero" que se movía en la escena capturada por el Optitrack. Así, mediante esta manipulación y control remoto indirecto, se podía realizar diversas pruebas de vuelo y estabilización.

3.2.5. Rutina de Comportamiento

Por cada ciclo de cómputo, dentro del Thread de manejo del cuadricoptero, se podía llamar a una función implementada llamada rutina() antes de realizar la acción de control. Esta función actualizaba el Set-Point de la posición espacial del cuadricoptero para que se modificaran de forma acorde los valores de roll, pitch y thrust, y lograr el desplazamiento deseado del aparato.

La función rutina() utilizaba las posiciones espaciales de todos los marcadores de la escena, capturados por el OptiTrack.

Bajo el control de esta rutina, el cuadricoptero era capaz de reconocer la ubicación de su punto de despegue y aterrizaje, como la posición de otros cuadricopteros, la posición de los objetos a ser trasladados, y la zona donde liberar dichos objetos. El set-point de control de posición del cuadricoptero era cambiado de forma dinámica para interactuar con su ambiente.

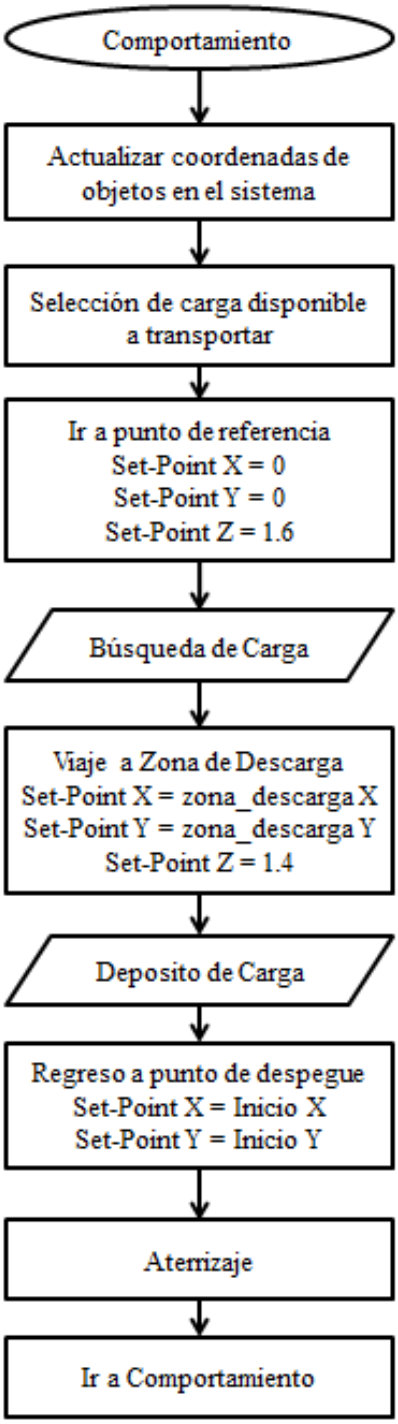


Figura 22: Diagrama de Flujo de rutina de comportamiento

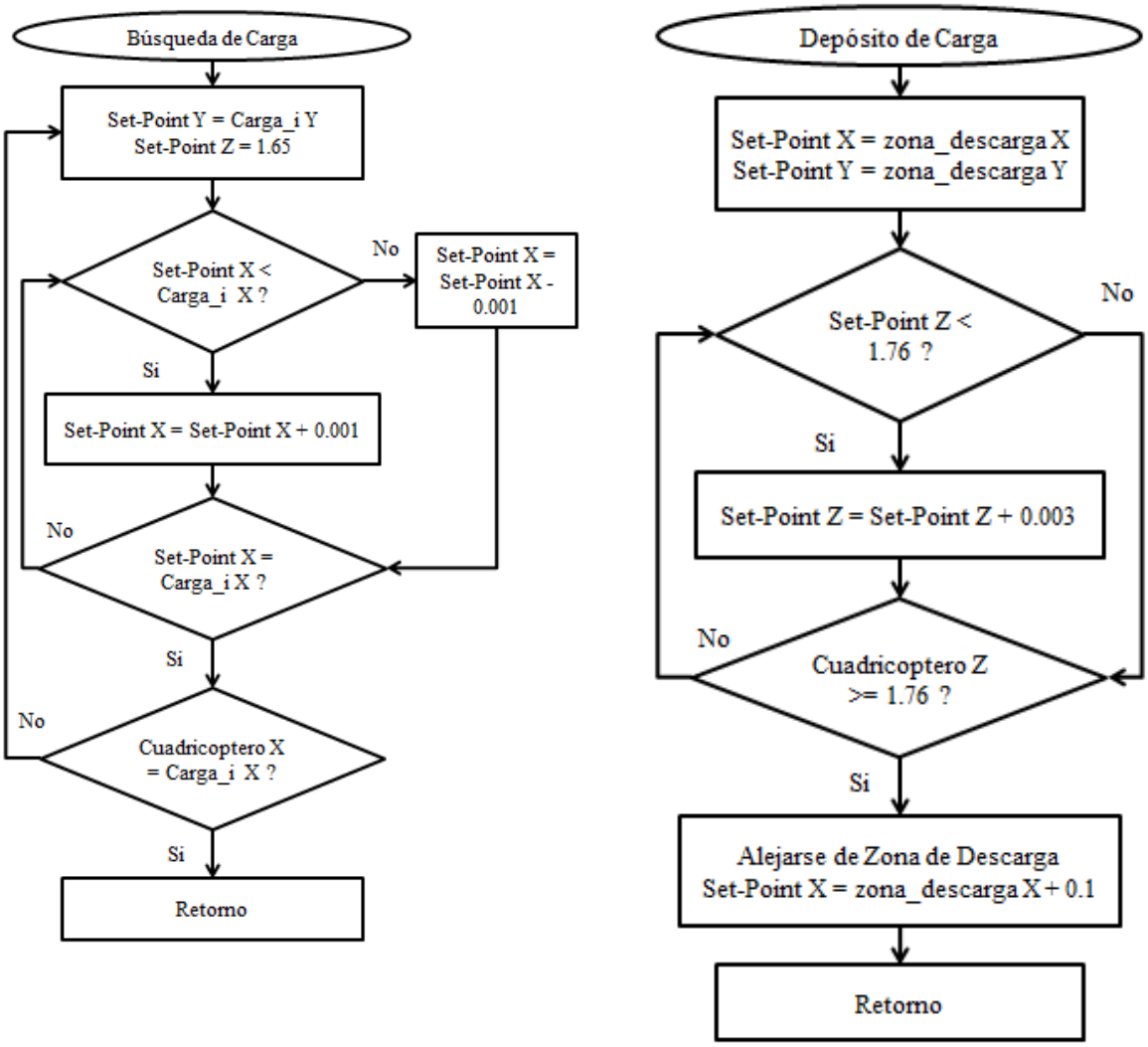


Figura 23: Diagramas de flujo de subrutinas de rutina de comportamiento

3.3. Transporte de Carga

Para realizar el transporte de carga, los cuadricopteros fueron instalados con un mecanismo de agarre simple de Ganchos de Rezón. El gancho estaba sujeto al centro de la base inferior del cuadricoptero mediante un alambre rígido de 10cm de largo, divido con 3 puntos de articulación. El mecanismo pesaba 2gr.

Para probar el mecanismo inicialmente se utilizaron pesos cuadrados de tamaño 2.5cm x 2.5cm con un aro de alambre para ser recogidos. Los pesos eran de diferentes valores: 1, 2 y 5gr.

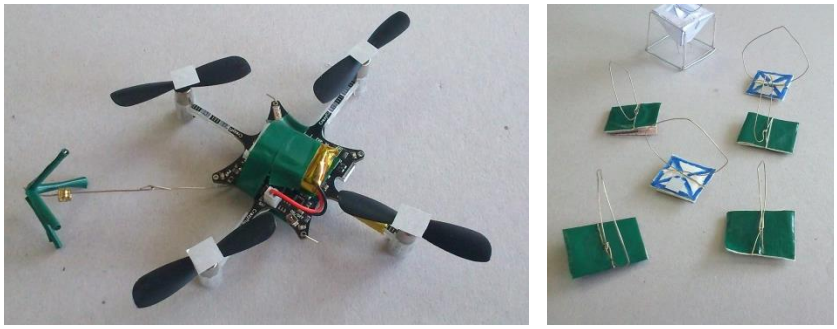


Figura 24: Gancho de Rezón y pesos de prueba

3.3.1 Idea y Concepto

Se probaron distintos diseños para realizar la tarea de transportar arena, granos de 2 a 3mm de grosor. Cada iteración del recipiente buscó resolver problemas tales como: la manipulación de su contenido a la hora de transportarlo y liberarlo, y su maniobrabilidad al quedar con un peso menor.

Dado el reducido peso adicional que puede transportar el cuadricoptero, y la imposibilidad de utilizar un mecanismo de agarre más avanzado con accionamiento controlado, se experimentó con varias maneras de combinar materiales livianos en la confección del recipiente para reducir su peso y su resistencia a las distintas corrientes de viento que genera el cuadricoptero.

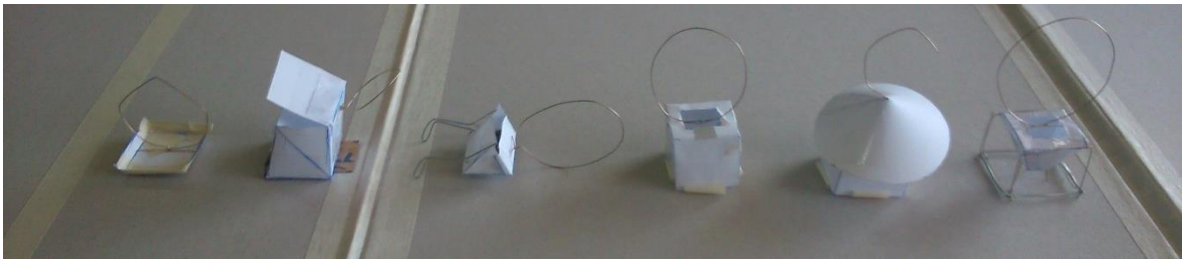


Figura 25: Evolución de los recipientes de carga

Los principales factores a considerar para mejorar el concepto del recipiente fueron su peso, forma y tamaño.

Al buscar un recipiente liviano y trasladarlo de forma inestable, colgando de un gancho de Rezón, se tomó en consideración el centro de masa del recipiente con y sin carga interna.

3.3.2. Evolución y versión final

La última versión pesó 2.8gr, el mayor peso de entre todas las iteraciones, debido al uso de más alambre para mantener el tamaño de la superficie de apoyo ya presente en las 2 últimas iteraciones. El centro de masa se mantenía más bajo de lo logrado anteriormente, debido al uso de un alambre más grueso en la cara inferior del armazón cubico.

Este recipiente también presentó la menor superficie de contacto al viento, minimizando la reacción oscilante ante el flujo de viento.

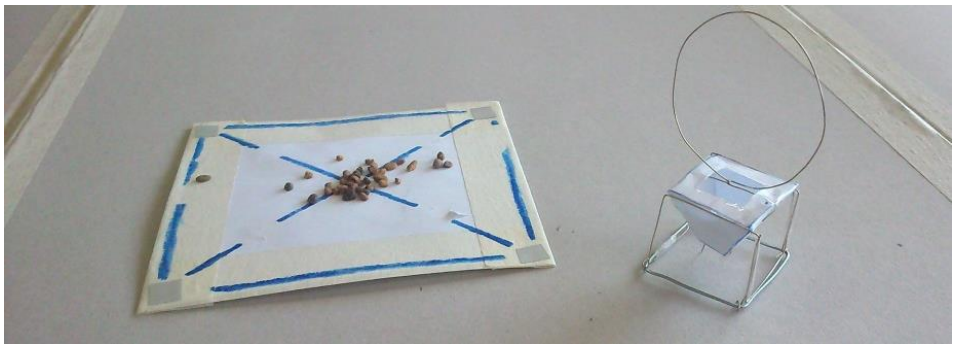


Figura 26: Recipiente actual

3.3.3. Mecanismo de funcionamiento

El cuerpo de papel del recipiente era una pirámide de base cuadrada truncada e invertida, abierta por arriba y abajo. En su interior, una pirámide de base cuadrada con el tamaño suficiente para no caer por la apertura de abajo de la pirámide mayor. El recipiente estaba rodeado de un esqueleto cubico de alambre. La pirámide interior tenía dos patas de alambre que sobresalen.

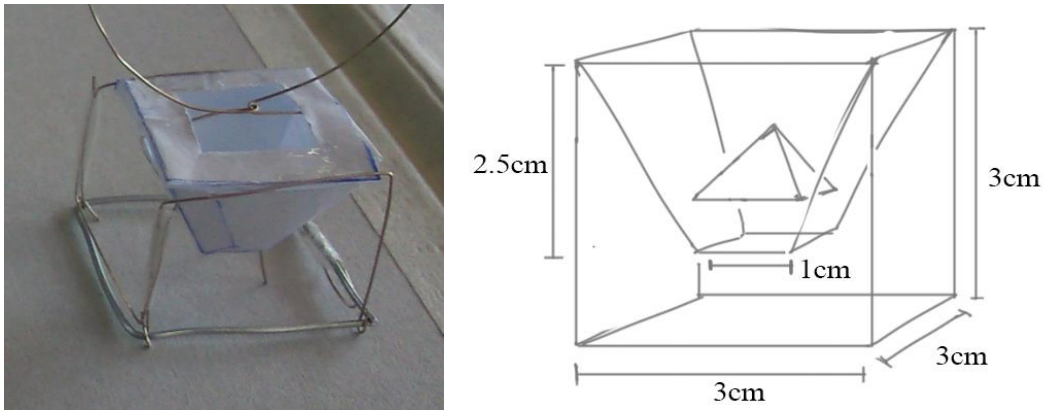


Figura 27: Esquema de recipiente

La pirámide al interior actuaba como tapón. Mientras el recipiente estuviera apoyado en un pedestal especial (A) o estuviera en aire (B), el tapón no se levantaría, impidiendo la caída de la arena. El mismo peso de la arena, y la forma del tapón, impedían que éste se moviera. Cuando el recipiente hiciera contacto con una superficie, el tapón se levantaría al hacer contacto primero, liberando la arena (C).

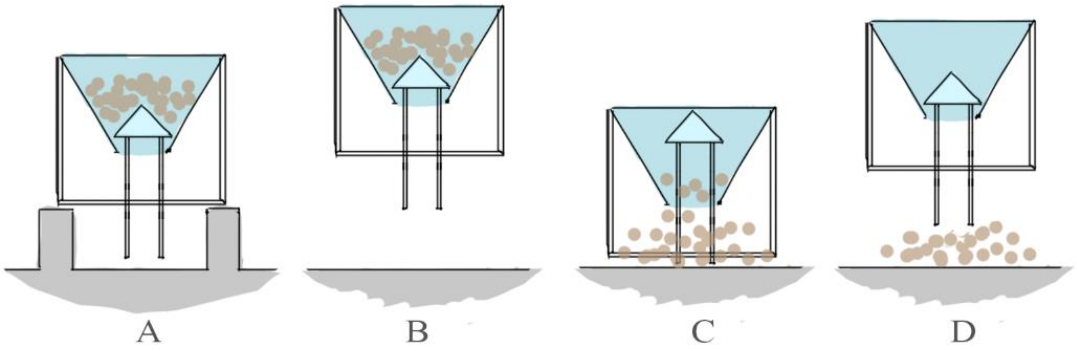


Figura 28: Funcionamiento del recipiente

4. RESULTADOS Y ANÁLISIS

En los capítulos anteriores se ha presentado la forma en que se abordaría el problema planteado y los conocimientos necesarios para su desarrollo. Habiendo realizado las pruebas necesarias, se exponen y discuten en este capítulo los resultados obtenidos en la realización de las actividades, junto con los análisis pertinentes.

Se posicionó un pliego cartón piedra plegable como superficie de pruebas de 110cm x 160cm, para expandir y aprovechar mejor la superficie de la mesa posicionada debajo del OptiTrack. Esta superficie es opaca y ayuda a la vista de los marcadores por las cámaras.

En algunas pruebas, se comparó el desempeño de los Crazyflie con un cuadricoptero comercial actual de similares prestaciones y características, el cuadricoptero Parrot MiniDrone Rolling Spider, y a otros proyectos similares dentro del Estado del Arte del campo.

4.1. Pruebas de estado de conexión

Se sometieron los cuadricopteros a Pruebas de distancia para medir el estado de su conexión, al punto de perder su comunicación con el programa cliente, obteniendo perdida de paquetes.

Sólo se mantuvieron encendidos, sin uso de sus motores, alejándolos de la fuente de comunicación de forma manual. Se usó el ancho de banda predefinido de 250kbit/s. Se utilizaron sus baterías con máxima carga.

	Rango de Crazyflie A [m]	Rango de Crazyflie B [m]
Exterior	75	30
Laboratorio y 5°Piso	11	1.5

El cuadricoptero B muestra un desempeño reducido de forma consistente con respecto al cuadricoptero A. Pierde fácilmente el link de comunicación ante alguna interferencia. Se detectó que su Chip de Antena 2.4-CHP-X tendría una falla de producción [6], ya que tampoco presenta mejoría al utilizar canales distintos para la comunicación.

Se repitieron las pruebas, pero cambiando el ancho de banda del cuadricoptero B a 2Mbit/s.

	Rango de Crazyflie A [m]	Rango de Crazyflie B [m]
Exterior	67	31
Laboratorio y 5°Piso	12	7

El rango del cuadricoptero B mejoró de forma considerable en la zona de interior, donde se presume la existencia de varias fuentes de interferencia. Al aumentar la tasa, la información permanece menos tiempo en el aire por lo que la probabilidad de que se provoque una interferencia disminuye.

El rango de radio depende ampliamente de varios factores, como la presencia de paredes u obstáculos en el ambiente, interferencia entre señales similares como antenas WiFi, Bluetooth, e incluso variaciones en el chip de antena de uno a otro Crazyflie.

En comparación, el cuadricoptero Parrot Rolling Spider puede volar en un rango promedio consistente de 20 metros tanto en interiores como exteriores. La conexión y manipulación del aparato se realiza a través de una aplicación para Smartphone o Tablet vía Bluetooth. El tipo de conexión del Crazyflie se beneficia de su tecnología para su uso en exteriores. En interiores, la diferencia en desempeño de ambos cuadricopteros es discutible.

4.2. Pruebas de Vuelo Iniciales

Thrust es un valor entero que va en el rango de 10001 (mínima potencia) a 60000 (potencia máxima). El envío de comandos puede realizarse a una tasa mínima de dos comandos por segundo, para prevenir que el cuadricoptero se desconecte por ausencia de un Set-Point de control.

El envío de comandos a una tasa aproximada de 100 comandos por segundo, es decir uno cada 10ms, permitió una respuesta fluida del cuadricoptero, sin estresar el rendimiento del sistema. Dada la tasa de captura del OptiTrack de 120 frames por segundo, la tasa de envío de comandos no necesita ser mayor.

4.2.1.Evolución del Control PID

Las primeras pruebas consideraron el siguiente desplazamiento a controlar, buscando una estabilización frente un punto dado (con el Set-Point).

	X [cm]	Y [cm]	Z [cm]
Set-Point	0	0	14
Posición Inicial	-37	32	14

Inicialmente, se consideró utilizar un único control PID para roll y pitch, el posicionamiento horizontal del cuadricoptero (ejes X e Y).

	Kp	Ki	Kd
Roll (Eje X)	18	0.5	2
Pitch (Eje Y)	18	0.5	2
Thrust (Eje Z)	500	0	5000

Los resultados obtenidos a continuación en la Figura 29.

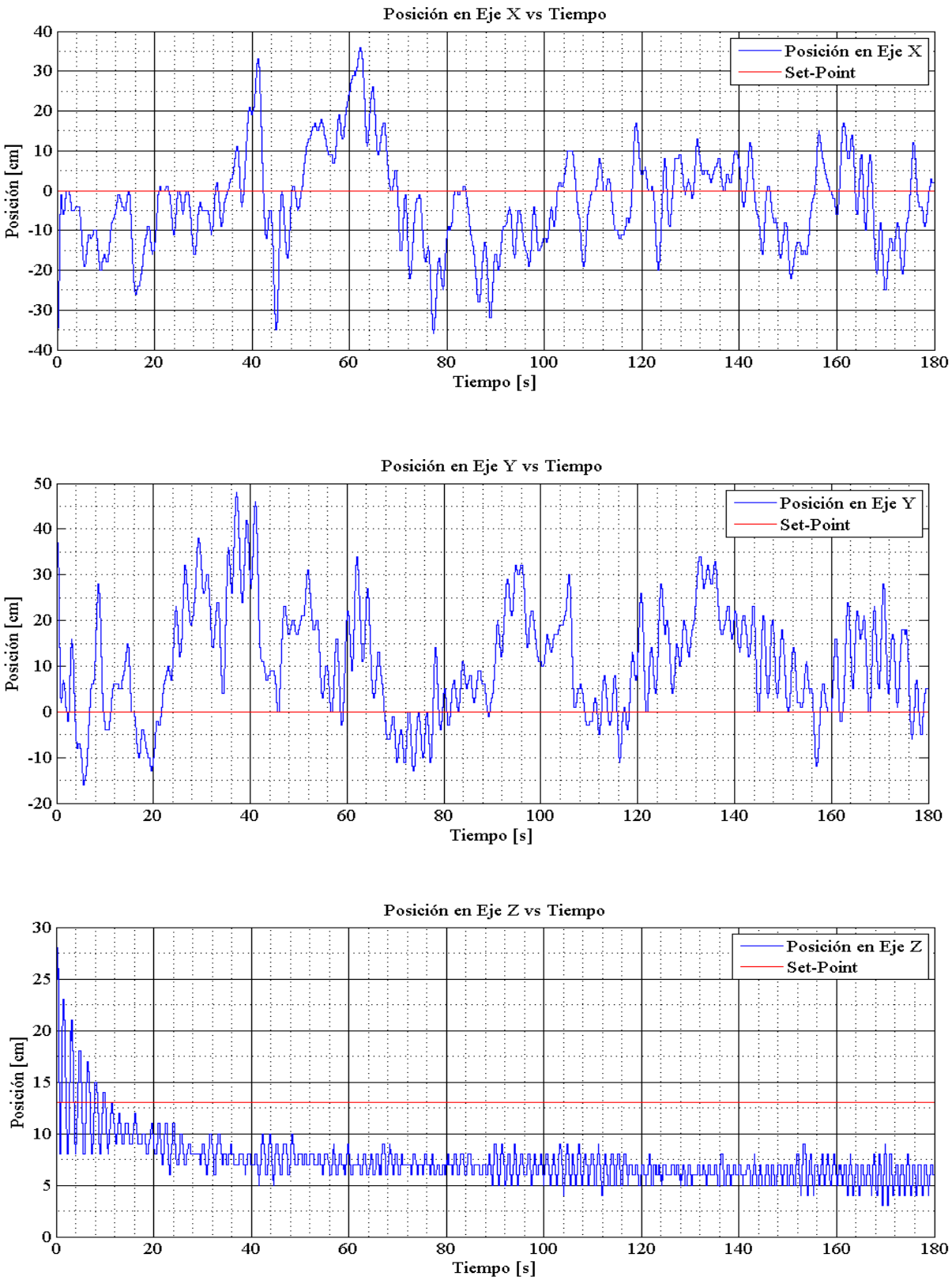


Figura 29: Gráficos de posicionamiento en X, Y y Z en inicio

El control PID para roll y pitch presentó grandes oscilaciones siendo de movimientos muy inestables, pero no divergió con el tiempo del Set-Point requerido.

El control PID para thrust presentó pequeñas oscilaciones en torno al Set-Point. Realizando una regresión lineal sobre la curva se aprecia la generación de un Error Acumulado. Conforme el Error Acumulado aumentaba, las oscilaciones se vuelven más notorias al alejarse del Set-Point.

Se iteró el cálculo de las ganancias PID, para mejorar la estabilización en torno al Set-Point.

	X [cm]	Y [cm]	Z [cm]
Set-Point	0	0	20
Posición Inicial	-33	32	26

	Kp	Ki	Kd
Roll (Eje X)	30	0.5	2

Los resultados obtenidos a continuación en la Figura 30.

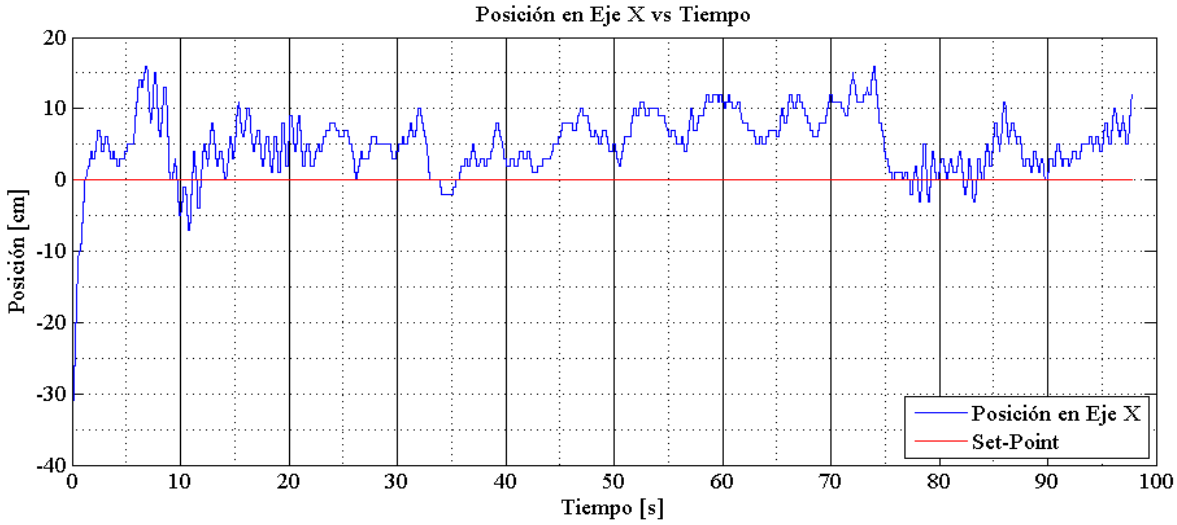


Figura 30: Grafico de iteración del control de posición en X

Se apreció un claro overshoot del control PID para roll. La ganancia Integral estaba demasiado alta en comparación a la Derivativa.

Se confirmó que el comportamiento de roll y pitch no era proporcional entre sí. Cada uno ameritaba su propio set de ganancias de control PID.

En esta iteración se probó dar el Set-Point al sistema de control 1 segundo después de comenzar el vuelo.

	X [cm]	Y [cm]	Z [cm]
Set-Point	0	0	20
Posición Inicial	-28	30	14

	Kp	Ki	Kd
Pitch (Eje Y)	35	0.7	5

Los resultados obtenidos a continuación en la Figura 31.

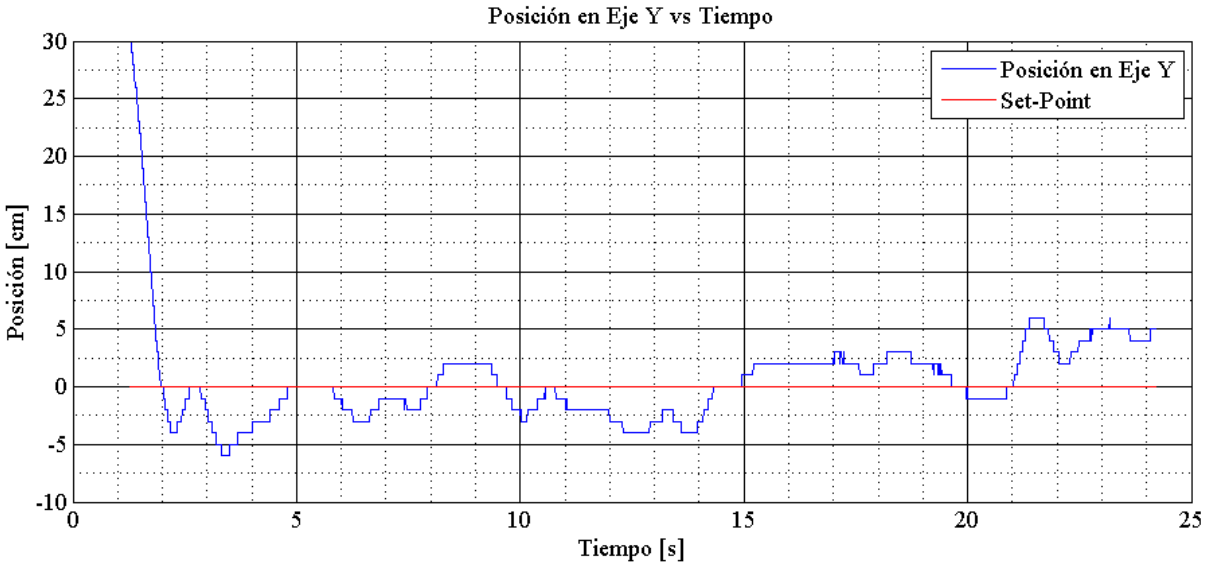


Figura 31: Grafico de iteración del control de posición en Y

Se agregó la ganancia Integral para compensar el Error Acumulado que se aprecia en el control PID de Thrust.

	X [cm]	Y [cm]	Z [cm]
Set-Point	0	0	20
Posición Inicial	-31	33	18

	Kp	Ki	Kd
Thrust (Eje Z)	500	1	5000

Los resultados obtenidos a continuación en la Figura 32.

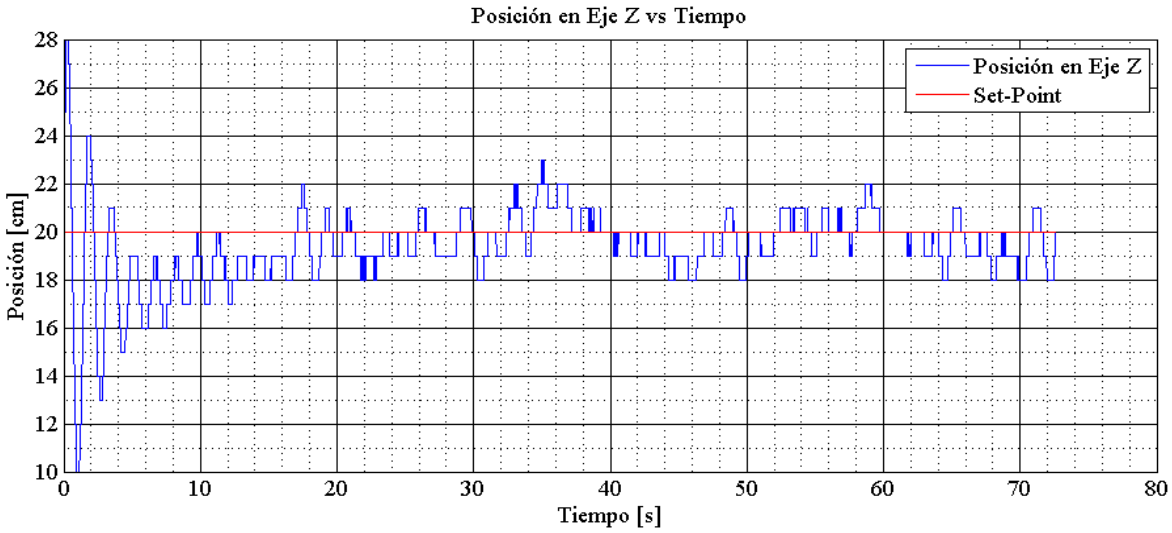


Figura 32: Grafico de iteración del control de posición en Z

El termino P termina ajustando la diferencia entre la altura objetivo y la altura del cuadricoptero. El término integral se encarga de la velocidad vertical.

La parte Integral del control compensa el Error Acumulado, que suple la caída de Voltaje del cuadricoptero por la descarga de la batería. Para compensar esa descarga continua, este valor se va acumulando a medida que el cuadricoptero siga volando. La ganancia utilizada se comportó mejor cuando el vuelo comienza con un 90-100% de carga de batería. A una carga de batería inicial más baja, el cuadricoptero presentaba pequeñas oscilaciones antes de converger en el Set-Point.

Finalmente el control PID fue pulido, hasta llegar a la siguiente iteración, la cual presenta las siguientes condiciones en su prueba.

	X [cm]	Y [cm]	Z [cm]
Set-Point	0	0	20
Posición Inicial	-33	30	10

El cuadricoptero fue caracterizado de mejor manera, apreciando que las ganancias de control para Roll, Pitch y Thrust deben ser independientes y diferentes.

	Kp	Ki	Kd
Roll (Eje X)	30	0.7	5
Pitch (Eje Y)	35	0.7	5
Thrust (Eje Z)	500	1	5000

Los resultados obtenidos a continuación en la Figura 33 y la Figura 34.

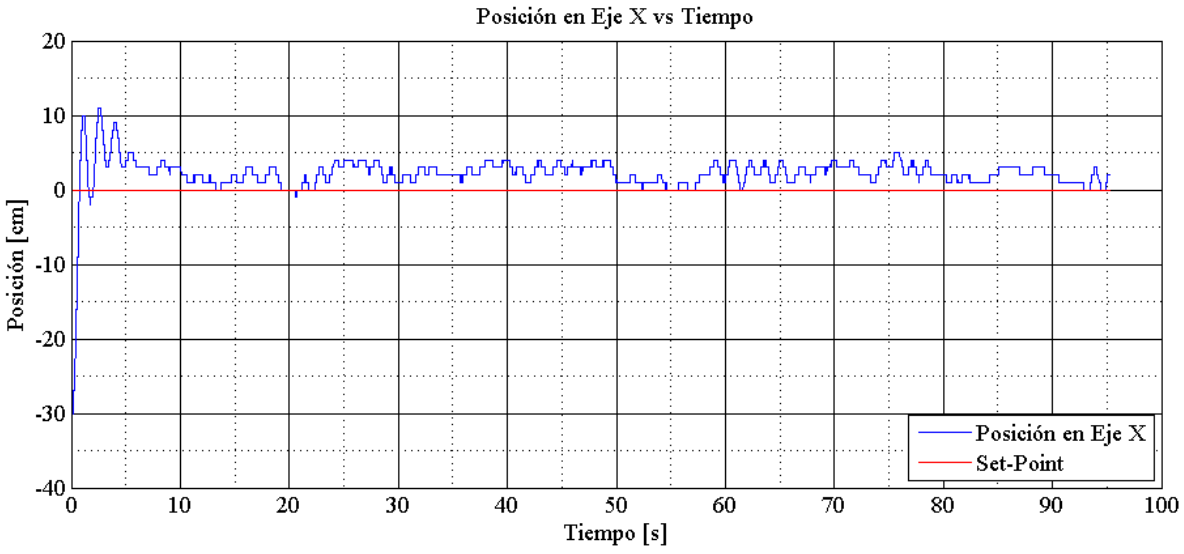


Figura 33: Grafico del control de posición en X

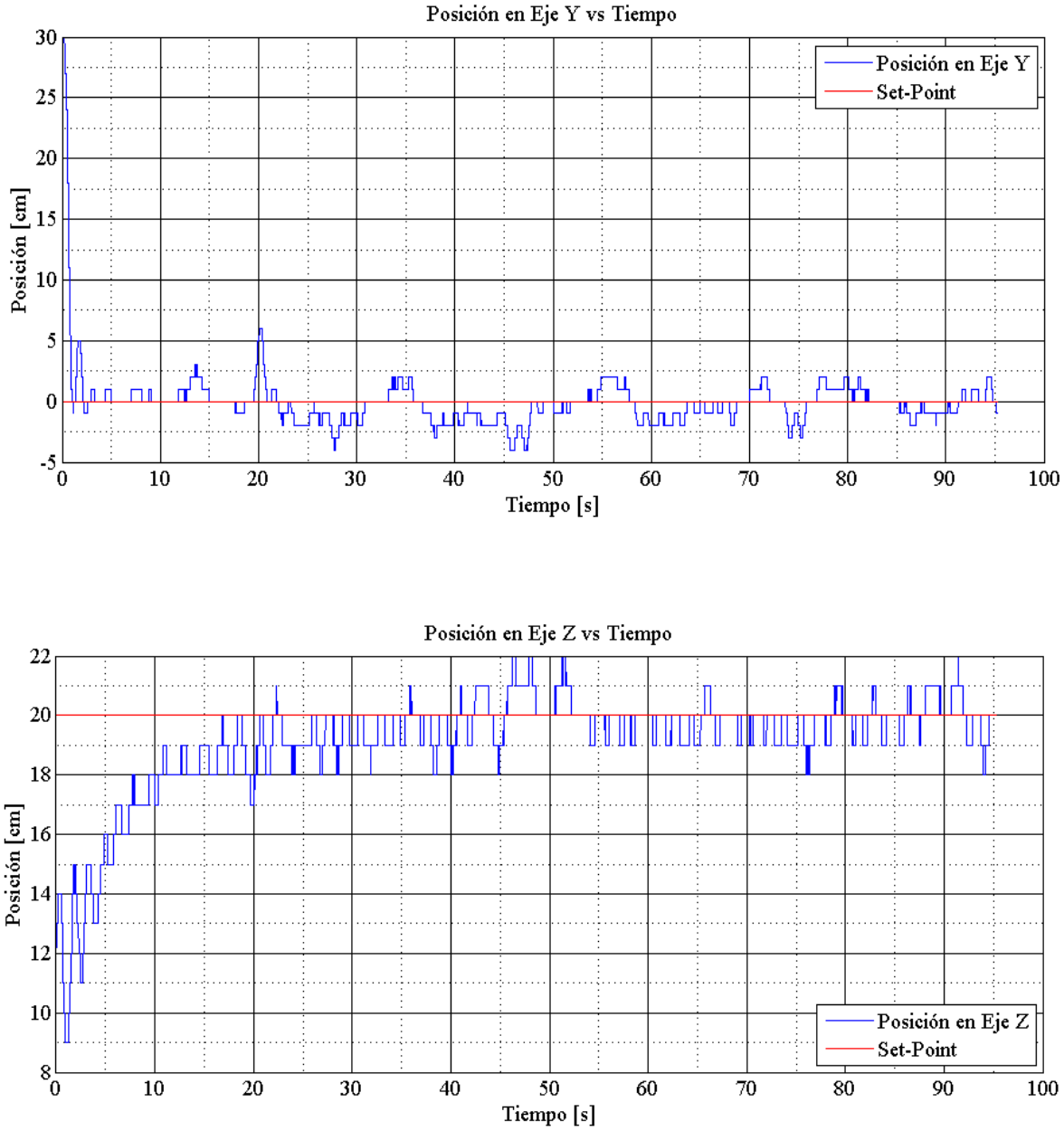


Figura 34: Gráficos del control de posición en Y y Z

Cabe destacar la importancia del correcto funcionamiento y configuración del OptiTrack, dado que el control PID sobre los cuadricopteros depende fundamentalmente de él para el lazo cerrado.

En los siguientes tres gráficos se demarcó con un asterisco verde (∗) los momentos en los que el OptiTrack no fue capaz de detectar correctamente los marcadores del Cuerpo Rígido de un Crazyflie, con lo que el feedback se hace erróneo, dado un error promedio superior a un umbral permitido.

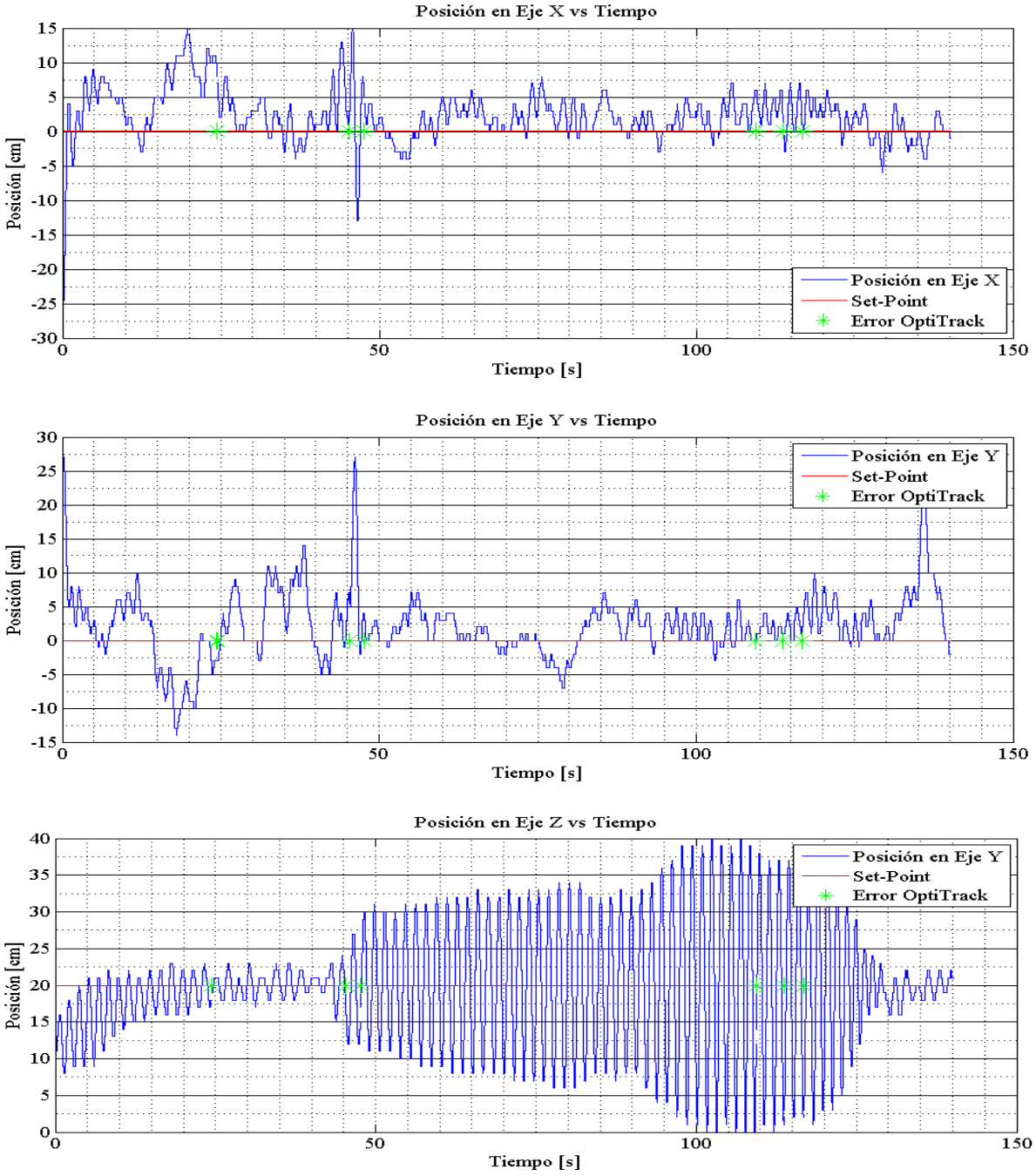


Figura 35: Gráficos de control con falla de identificación de marcadores

4.3. Pruebas de Descarga de Batería

El cuadricoptero fue sometido a distintas pruebas de descarga. Para esto, se utilizó una batería de 170mAh de capacidad, con 3.7V (1 Celda) a máxima carga, la cual toma en promedio 20 minutos en recargar por completo.

El cuadricoptero pudo permanecer encendido, sin movimiento de los motores con un Thrust=0, por 30 minutos. Al utilizar la potencia mínima con la que empieza a girar los motores, un Thrust=10001, la cual no permite que el cuadricoptero despegue, el tiempo de descarga de la batería fue de 16 minutos.

Para condiciones de vuelo efectivo, en un estado de Hover, sin presencia de perturbaciones, el tiempo de vuelo se redujo a 7-8 minutos. Para vuelos con desplazamiento continuo el tiempo se vio reducido a 5-6 minutos.

La generación de mayores aceleraciones y cambios de direcciones bruscas generaron un decremento de tiempo de vuelo percibible.

En comparación, el cuadricoptero Parrot MiniDrone Rolling Spider presentaría tiempos de vuelo máximos de 8 minutos, promediando 7 minutos en vuelos de movilidad continua. Para levantar sus 55gr. de peso, el cuadricoptero utiliza una batería LiPo de 550mAh, la cual toma 90 minutos para recargar. El tiempo de vuelo estimado sería ligeramente superior al del Crazyflie, aun así manteniéndose dentro del promedio para cuadricopteros de esta envergadura , pero con la enorme desventaja de un tiempo de recarga muy superior. El tiempo de autonomía de cuadricopteros miniatura es reducido y similar entre ellos.

4.4. Pruebas de seguimiento y estabilización

El cuadricoptero fue expuesto a unas pruebas donde el Set-Point es cambiado a lo largo del tiempo, para evaluar la respuesta del control PID. Se utilizaron funciones de decisión que alteraban el Set-Point acorde a una franja de tiempo estipulada. Los resultados obtenidos son positivos (Figura 36).

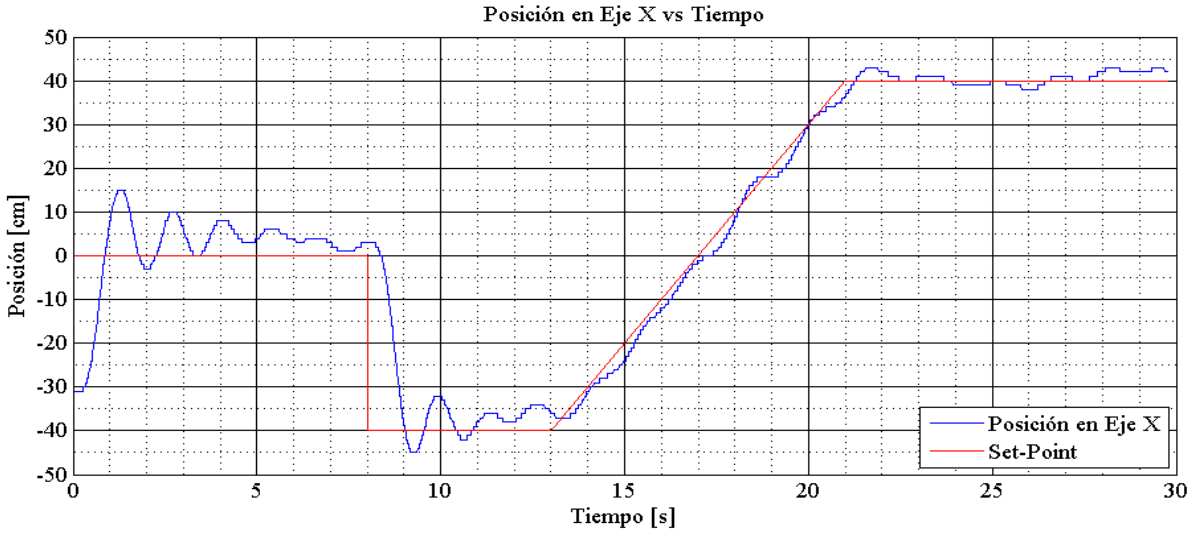


Figura 36: Grafico de control de posición en eje X con Set-Point variable

También se sometió el cuadricoptero a perturbaciones, como flujos esporádicos de viento y alteración de su posicionamiento por terceros, realizando un retorno a su lugar determinado por el Set-Point de posición.

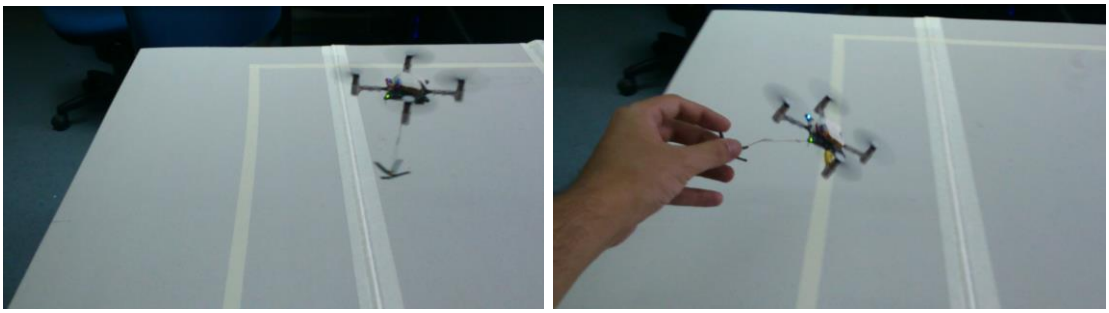


Figura 37: Perturbaciones

Estas fueron las bases para crear la “Rutina de Seguimiento”, como una función complementaria al fin de este proyecto, para probar el movimiento del cuadricoptero ante un Set-Point variable en el tiempo.

El Set-Point en este caso fue dado por la posición de los marcadores de un objeto puntero en la escena. El objeto Puntero fue tomado por un usuario para realizar un control remoto directo del cuadricoptero, realizando movimientos con la mano los cuales son imitados por el aparato.

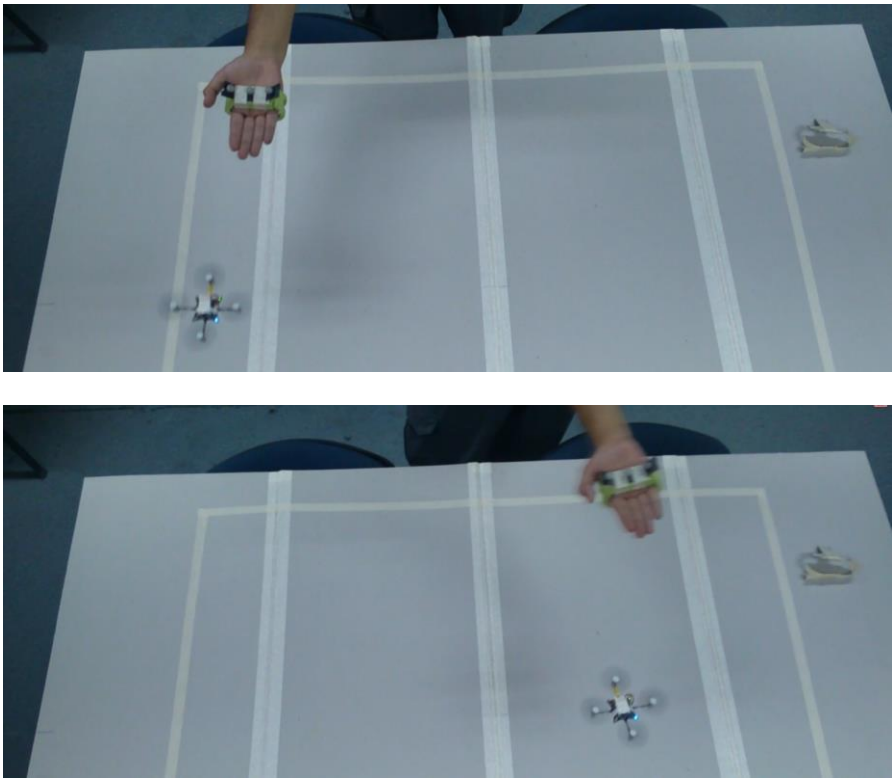


Figura 38: Control remoto con Rutina de seguimiento

Esta Rutina adicional facilitó la mejora de la “Rutina de Comportamiento” al mostrar un acercamiento intuitivo al comportamiento de respuesta del cuadricoptero ante este tipo de pruebas adicionales.

Además, se realizaron las siguientes comparaciones. Se hizo una evaluación del control horizontal (para pitch o roll) frente a un cambio en escalón de la posición demandada. Se comparó el comportamiento frente al control PID horizontal del MikroKopter [18], un cuadricoptero mediano de código abierto.

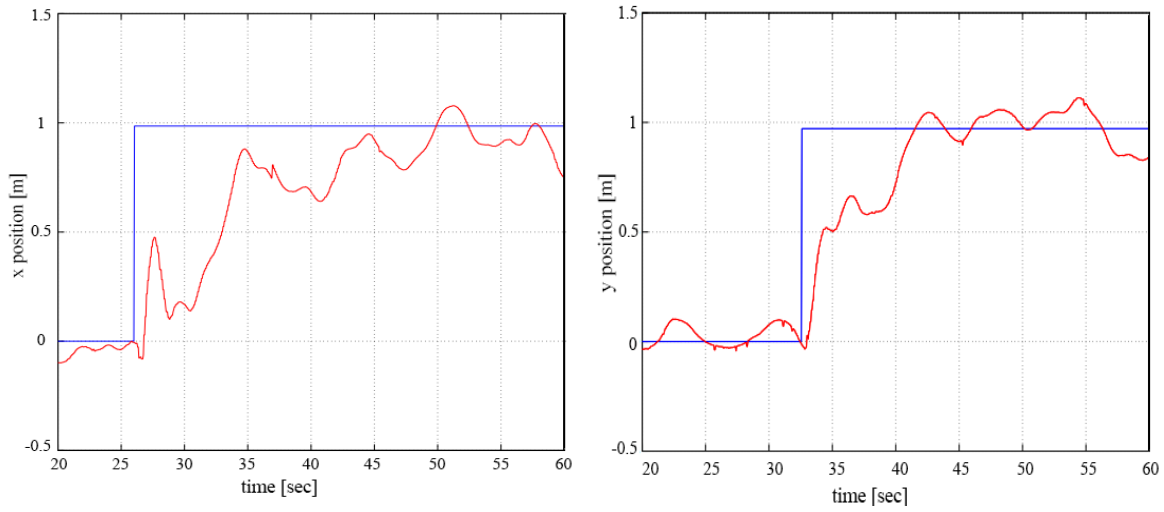


Figura 39: Respuesta obtenida para eje X e Y ante un cambio instantáneo de la posición requerida.

La respuesta obtenida para el eje X e Y ante un cambio instantáneo de la posición requerida, donde el color azul denota la posición demandada y el rojo es la posición medida por un láser escáner ICP.

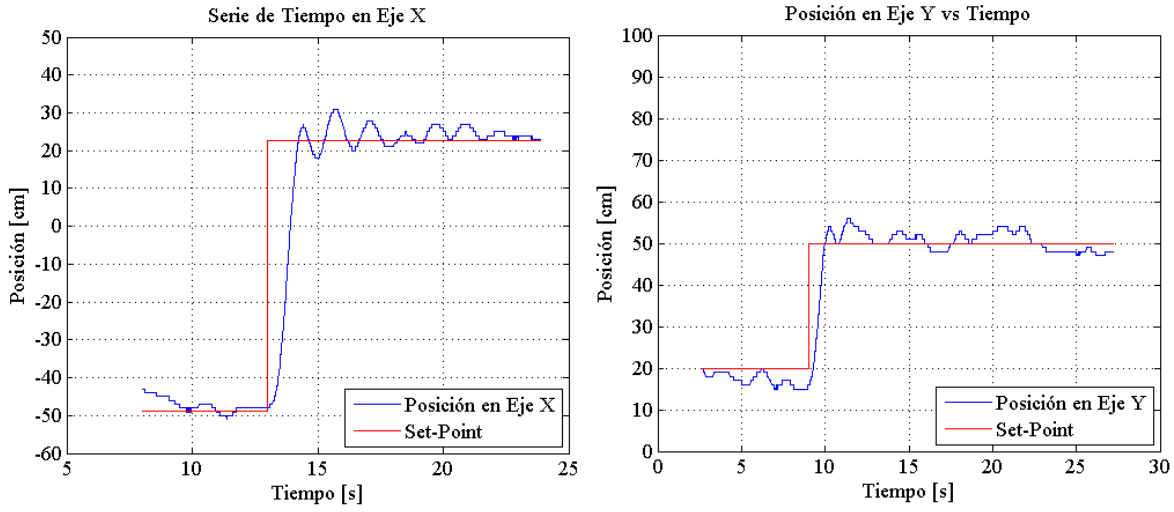


Figura 40: Comportamiento ante cambio instantáneo en posicionamiento horizontal implementado en Crazyflie

Se apreció una respuesta rápida frente a la presentada por el MikroKopter.

Otra evaluación consistente en la estabilización horizontal en torno a un punto fijo, se comparó el desempeño de un sistema de control PID simulado para el diseño de un cuadricoptero miniatura autónomo [19].

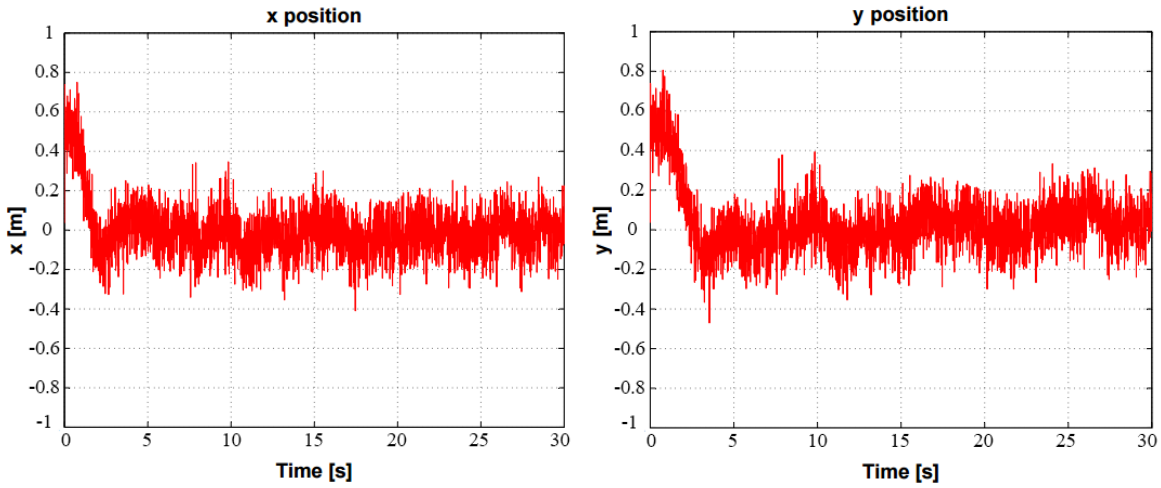


Figura 41: Estabilización horizontal cuadricoptero miniatura autónomo

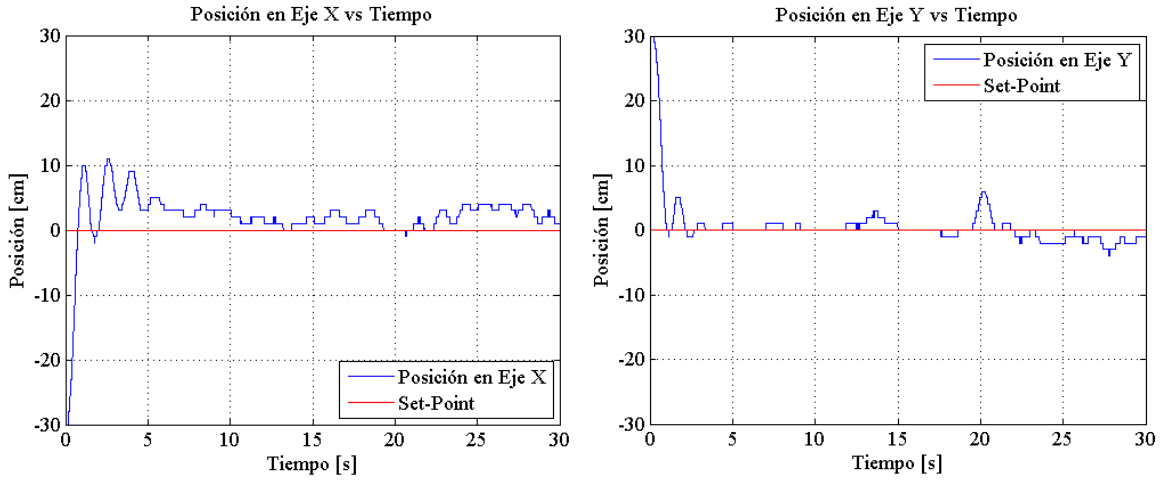


Figura 42: Estabilización horizontal implementada en Crazyflie

La implementación del trabajo presente mostro un tiempo de estabilización similar, pero se aprecia una menor amplitud en la oscilación de la señal.

Para el mismo proyecto, se evaluó el control PID vertical para el despegue y hover del cuadricoptero miniatura autónomo. En esta figura, se aprecia la simulación y la posteior implementación de aquel proyecto [20].

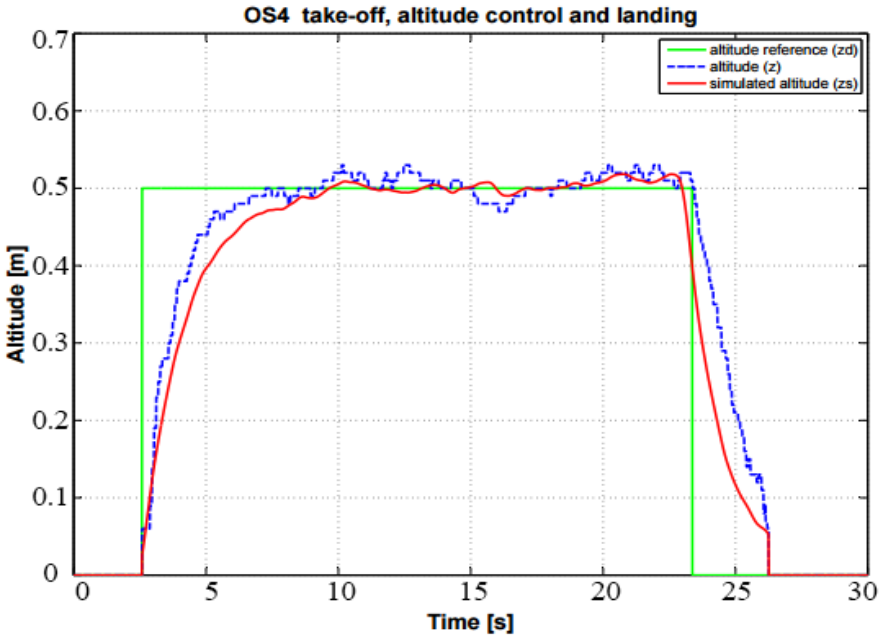


Figura 43: Despegue, control de altitud y aterrizaje simulado y real de cuadricoptero miniatura autónomo

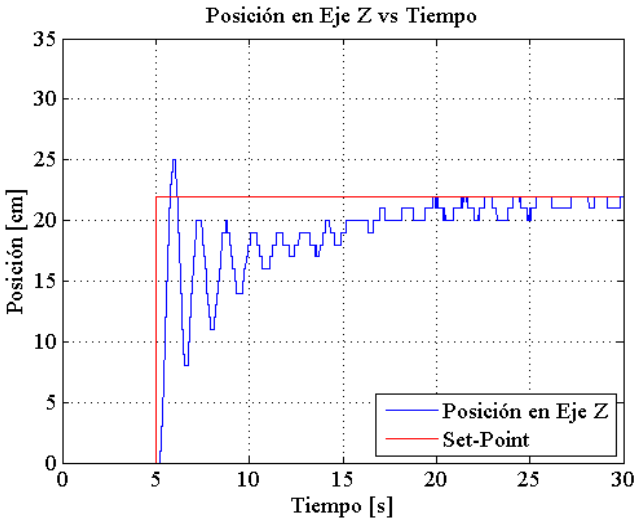


Figura 44: Despegue y control de altitud implementado en Crazyflie

La implementación del trabajo presente mostro un overshoot, junto con un tiempo de estabilización mayor (5 segundos adicionales).

4.5. Pruebas de Peso

Se realizaron pruebas de elevación con y sin cargas adicionales de peso de 1gr, 2.5gr, 5gr.

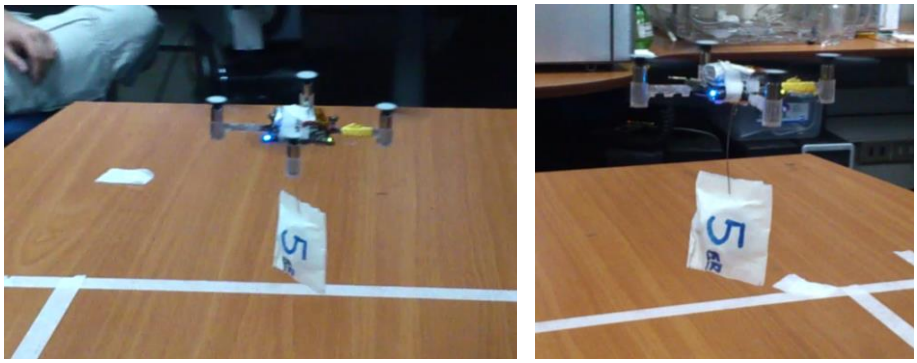


Figura 45: Levantamiento de peso

Las Pruebas de peso determinaron que el PID de thrust era muy lento en respuesta a aumentar la carga del cuadricoptero, demorando su elevación ante el incremento de peso. El Crazyflie pesa 19gr, sin el gancho de Rezón encima, por lo que el peso adicional afecta fuertemente su maniobrabilidad y desempeño.

Se incrementó la ganancia Derivativa del control PID de thrust, para incrementar la velocidad de respuesta.

	Kp	Ki	Kd
Thrust (Eje Z)	500	1	7500

El peso adicional máximo que puede sostener el cuadricoptero sin perjudicar su maniobrabilidad en exceso es de 8gr. Con un peso mayor, la elevación del Crazyflie se hace difícil, incluso ocupando thrust máximo, con una carga media de la batería.

El maniobrar con un peso de 8gr, redujo su tiempo de vuelo efectivo de forma drástica, a tan solo 2 minutos, por la potencia utilizada para ejecutar su desplazamiento de forma correcta.

Se presenta a continuación una prueba con un peso de 6gr.

	X [cm]	Y [cm]	Z [cm]
Set-Point	0	0	20
Posición Inicial	-29	29	2

	Kp	Ki	Kd
Roll (Eje X)	30	0.7	5
Pitch (Eje Y)	35	0.7	5
Thrust (Eje Z)	500	1	7500

Los resultados obtenidos a continuación en la Figura 46.

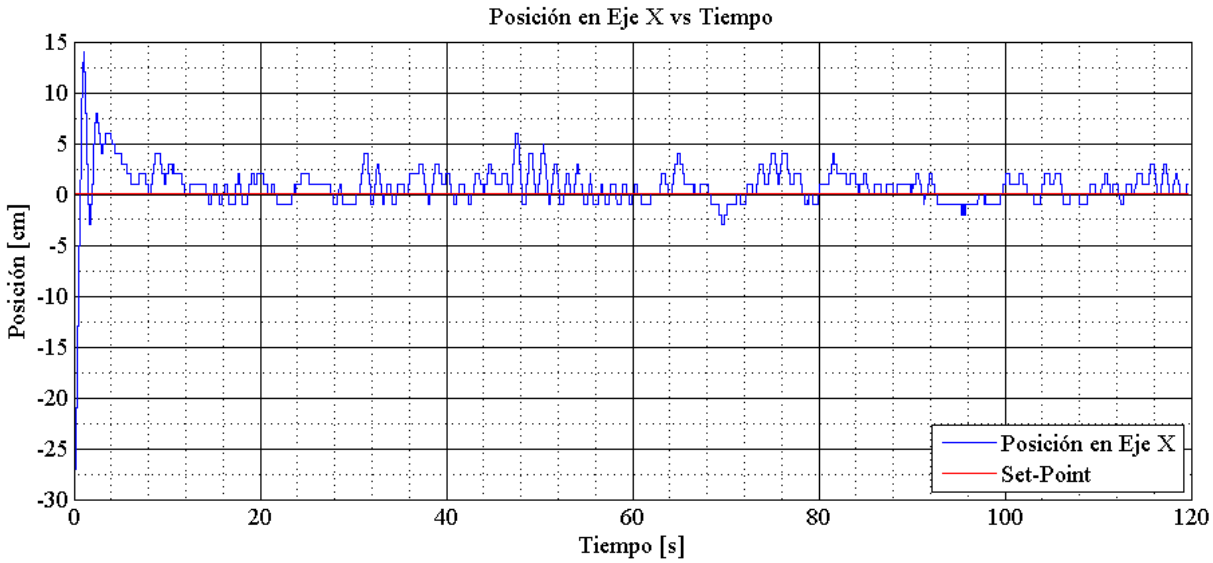


Figura 46: Grafico de control de posición en eje X al levantar peso

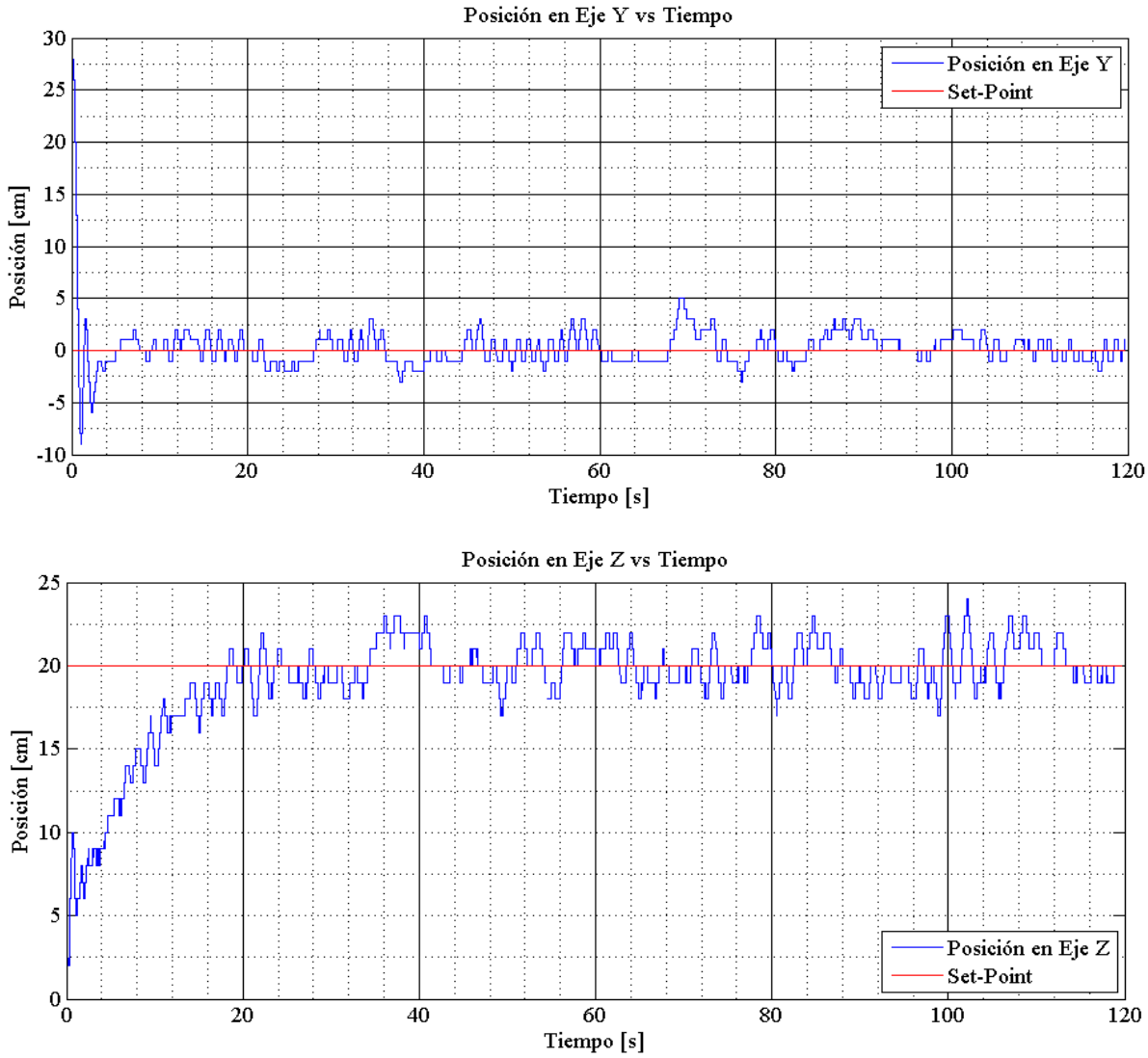


Figura 47: Gráficos de control de posición en Y y Z al levantar peso

El comportamiento del cuadricoptero fue adecuado, presentando pequeños overshoots en el control de pitch y roll. Para ello se decidió incrementar la acción de control sobre el cuadricoptero llegando al PID actual.

	Kp	Ki	Kd
Roll (Eje X)	33	0.8	5
Pitch (Eje Y)	38	0.95	5
Thrust (Eje Z)	500	1	7500

4.6. Pruebas de Navegación

Se realizaron numerosas pruebas de traslado de carga de forma automatizada, utilizando la "Rutina de Comportamiento" implementada. Se trasladaron cargas de pesos de 3gr y recipientes con arena con un peso total de 5gr. El lugar de descarga es un cuadrado de 14cm x 14cm, con marcadores para su reconocimiento por el OptiTrack.

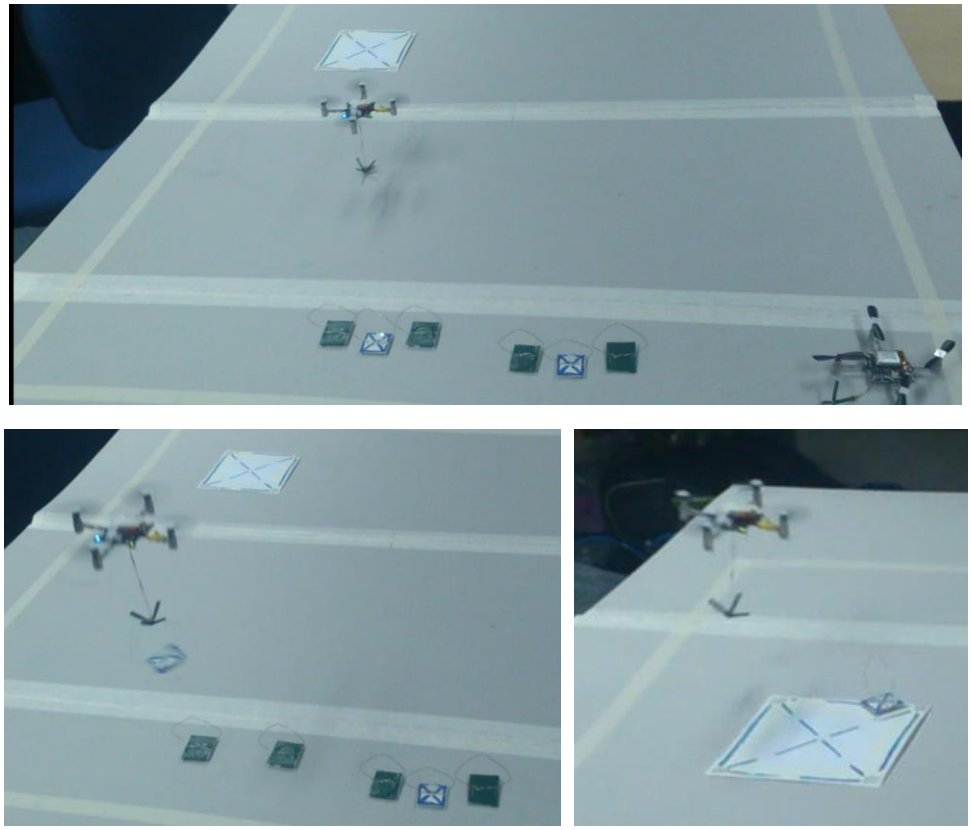


Figura 48: Rutina de comportamiento con pesos

El traslado de una carga toma la siguiente cantidad de tiempo: Ir al punto de referencia toma 4 segundos. La búsqueda del material desde el punto de referencia toma de 7 a 10 segundos. El viaje a la zona de descarga es de 5 segundos. Descender y depositar la carga toma 10 a12 segundos. Regreso al punto de partida toma 5 segundos. El aterrizaje toma de 2 a 3 segundos.

Para una prueba en particular, la navegación automatizada de un cuadricoptero siguió el siguiente desplazamiento en el tiempo.

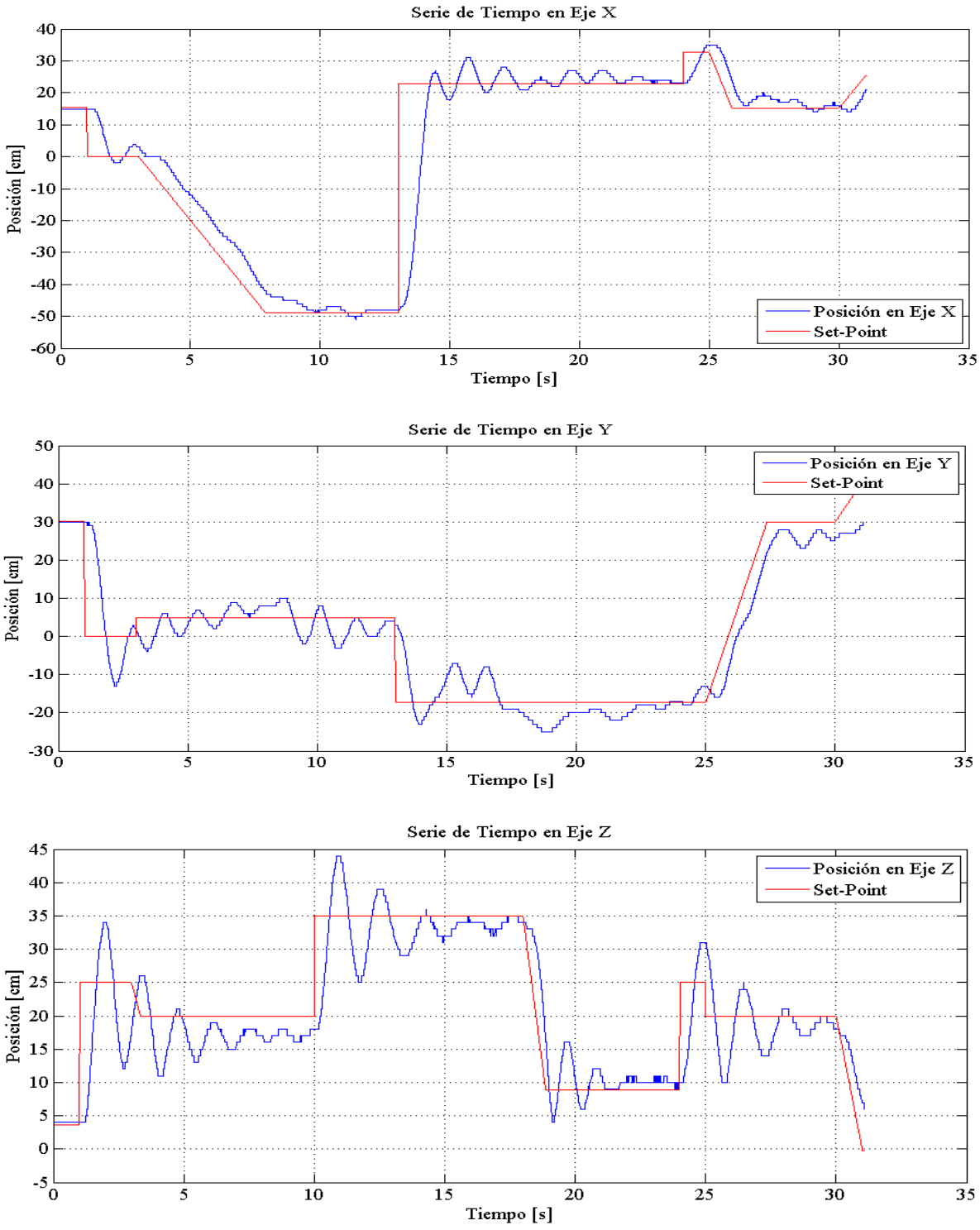


Figura 49: Desplazamiento en Prueba de Navegación

Se desarrolló la siguiente trayectoria en el espacio.

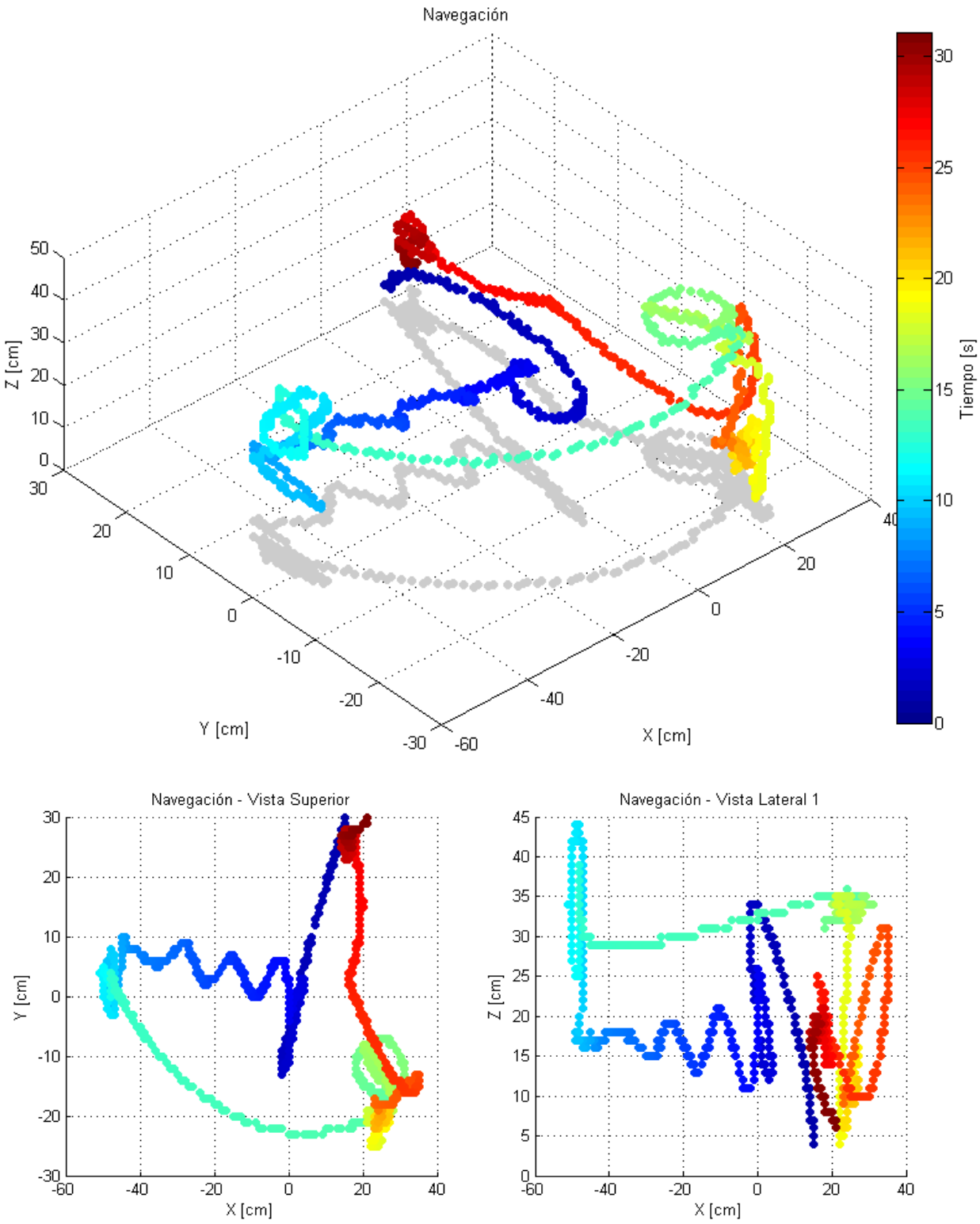


Figura 50: Navegación en vista isométrica, vista superior y vista lateral

El siguiente traslado lo realizaría un segundo cuadricoptero que comienza cuando el primero fuera a aterrizar y despejar el campo aéreo (Figura 51 B). El segundo cuadricoptero despegaría aproximadamente a los 27 segundos de vuelo del primero, cuando éste ya estuviera por aterrizar.

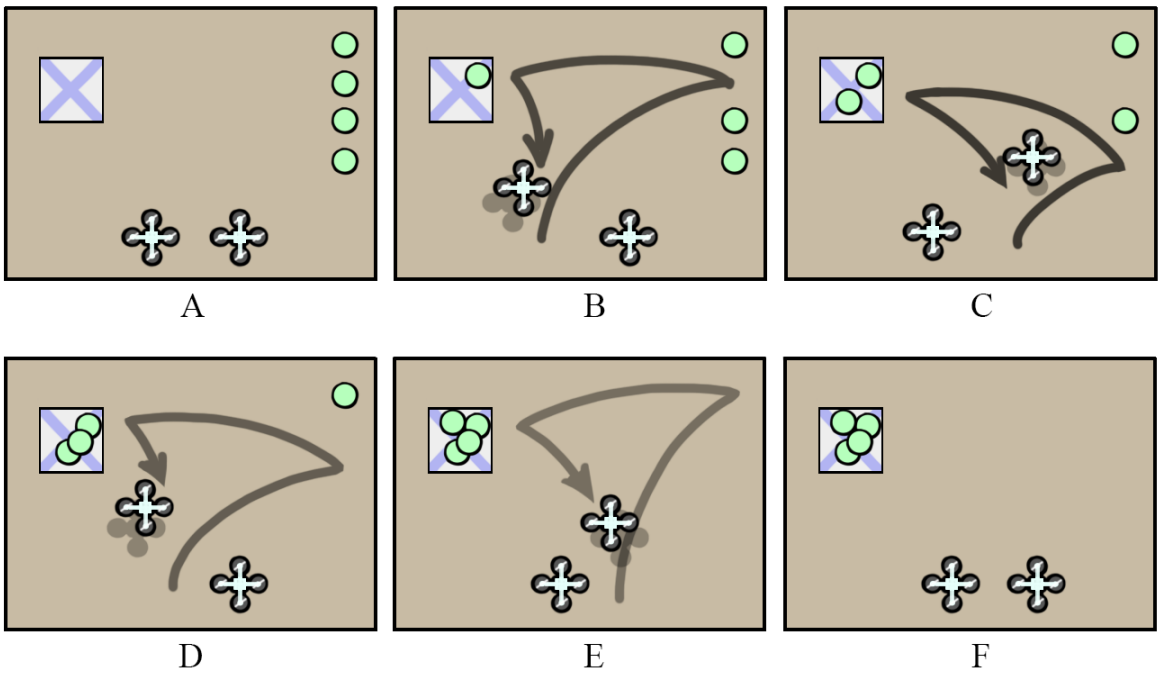


Figura 51: Esquema genérico de relevo en la Navegación

El segundo cuadricoptero repetiría la misma rutina que desarrolló el cuadricoptero anterior, limitándose a buscar los pesos que aún no han sido trasladados, ie. Están alejados de la zona de descarga (Figura 51 C). Para hacer eso, antes de elevarse se chequea la posición actualizada de todos los objetos del ambiente, incluida la ubicación de la zona de descarga, la posición de todas las cargas y el punto de aterrizaje del primer cuadricoptero.

El vuelo de los cuadricopteros se repite en sucesión hasta que todas las cargas han sido depositadas en la zona de descarga, o se presenta la desconexión de alguno de los aparatos por descarga de batería, interferencia en la conexión, o la interrupción de un usuario.

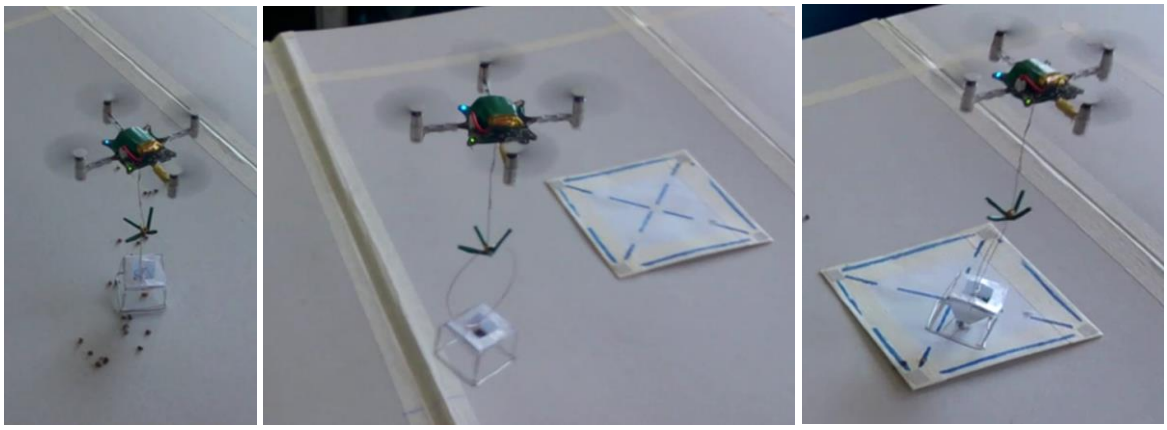


Figura 52: Rutina de comportamiento con recipiente de arena

Para trasladar los recipientes con arena, se utilizó la misma rutina, con la excepción de que en la zona de descarga se deposita solo la arena y el recipiente fue llevado a un punto aleatorio cercano para desecharlo y luego aterrizar. Esta acción adicional lleva de 8 a 10 segundos extras, enterando 40 segundos en total por vuelo de traslado.

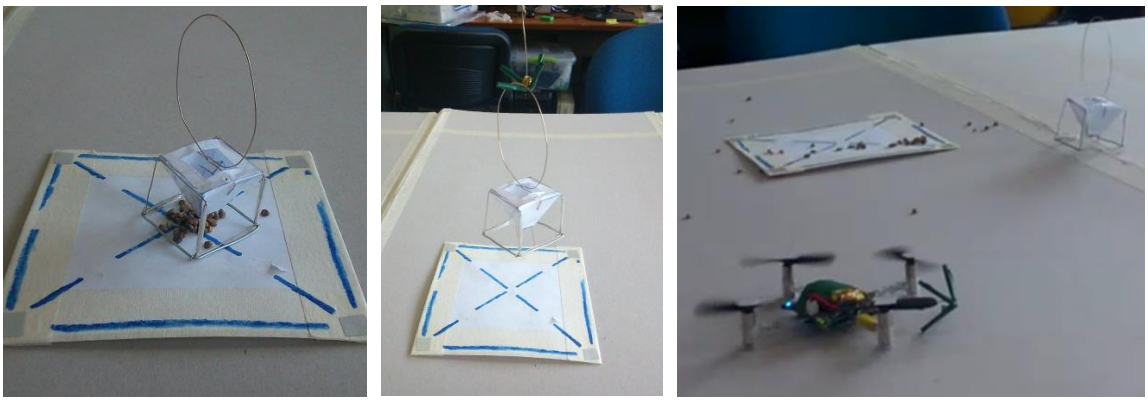


Figura 53: Recipiente de arena y zona de descarga

Por cada iteración del viaje, se liberaron de 2.2 a 2.6gr de arena transportada en un recipiente que pesa aproximadamente 2.8gr. En 2.5gr de arena existen aproximadamente de 30 a 40 piedras, de un tamaño de 2 a 3mm cada una.

El montículo de arena depositada cubrió en promedio 25cm^2 de superficie, dependiendo de la velocidad con que se bajara el recipiente a la zona de descarga.

A medida que avanzó la descarga de arena, se apreció la tendencia de la acumulación en torno al centro de la zona de descarga, coordenadas precisas para el descenso. El desempeño presenta una disposición donde la densidad del depósito de arena asemeja una distribución normal, dada por la presencia de perturbaciones y pequeñas oscilaciones del cuadricoptero al descender.

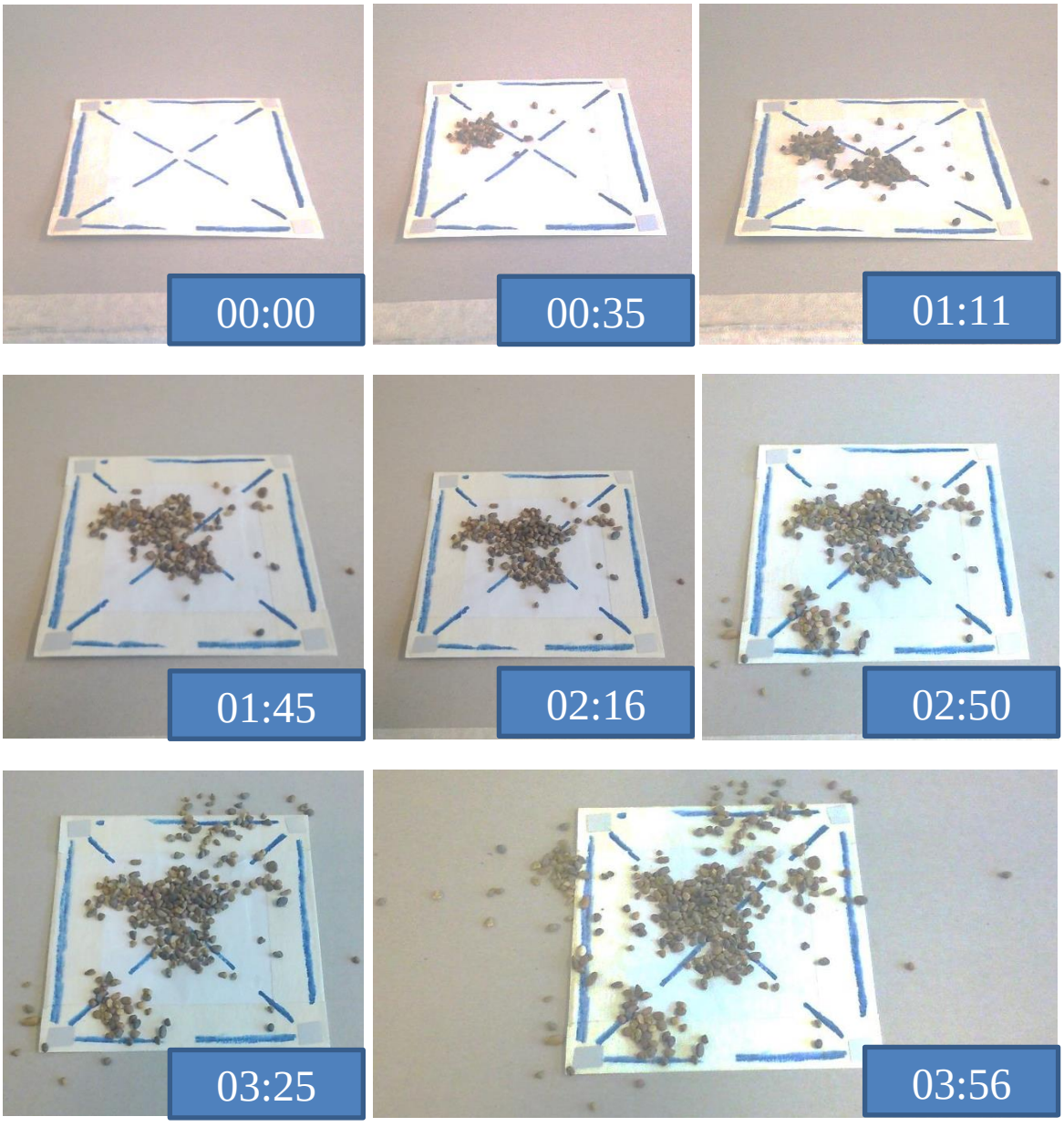


Figura 54: Acumulación de arena descargada en el tiempo

5. CONCLUSIONES Y RECOMENDACIONES

Los cuadricopteros son drones cuya principal ventaja ante otro tipo de vehículos autónomos es su mayor maniobrabilidad, dado que pueden mantenerse suspendidos, quietos en el aire. Una de las motivaciones para la implementación de este proyecto está relacionada con la posibilidad de explotar esa cualidad estudiando una aplicación de transporte y montaje a pequeña escala.

Se implementó bajo la metodología propuesta un sistema automatizado de cuadricopteros de tamaño reducido, pudiendo trabajar bajo un sistema de relevo. Al actualizar sus tareas propias con la labor ya realizada por su par anterior, los cuadricopteros pueden sincronizarse y transportar pequeñas cantidades de material, arena en particular, sumando así a un resultado final conjunto que termina siendo de carácter aleatorio. Esta implementación buscó homologar y aplicar en cierta manera, haciendo caso a las limitaciones pertinentes, un concepto que se da en la naturaleza donde la unión del trabajo de más de un individuo, terminan por sumar sus resultados individuales pero complementarios.

Siendo aplicada la teoría de control, la automatización correcta del sistema permite un grado maniobrabilidad extra en los cuadricopteros miniatura que un cuadricoptero mayor no puede tener. Sus dimensiones menores permitieron movimientos rápidos y precisos en espacios de trabajo reducido, y la interacción con materiales livianos a transportar, sin ser mayormente perjudicados por las corrientes de aire que produce el sustento del aparato.

La unidad de sensado utilizada, el OptiTrack, es un sistema de captura profesional que realiza un trabajo completo, facilitando este aspecto del sistema. El hecho de utilizar únicamente este equipo como feedback del lazo cerrado en tiempo real, pudo incurrir fácilmente en una acumulación de error, dada su extensa configuración y la aparición de cualquier externalidad visual, como reflejos no deseados, materiales inadecuados, iluminación fluctuante, etc. Esto conllevo a usar un ambiente de pruebas estricto.

Los resultados obtenidos resaltan que el tiempo de vuelo efectivo de los cuadricopteros es un factor vital a la hora de decidir una buena aplicación. Los cuadricopteros miniatura pueden trabajar en el sistema planteado y ser puestos a prueba, realizando labores de carácter puntual con la finalidad de maximizar la eficiencia energética. Se recomienda estudiar el uso de una fuente recargable energía para los cuadricopteros. De esa forma una de las principales limitantes del sistema seria superada.

BIBLIOGRAFÍA

[1] Crazyflie control board Rev.F component placement top [en línea] <http://wiki.bitcraze.se/_media/projects:crazyflie:hardware:crazyflie_control_board_rev.f_-_component_placement_top_annotated.pdf>

[2] Crazyflie control board Rev.F component placement bottom [en línea] <http://wiki.bitcraze.se/_media/projects:crazyflie:hardware:crazyflie_control_board_rev.f_-_component_placement_bottom_annotated.pdf>

[3] The Crazyflie Python API [en línea] <http://wiki.bitcraze.se/doc:crazyflie:api:python:index>

[4] CRTP Overview [en línea] <http://wiki.bitcraze.se/doc:crazyflie:api:python:index>

[5] Bitcraze Wiki [en línea] <http://wiki.bitcraze.se/projects:crazyflie:index>

[6] Antenna Factor – Ultra compact Chip Antenna Data Guide [en línea] <http://www.linxtechnologies.com/resources/data-guides/ant-xxx-chp-x.pdf>

[7] NatNet API User's Guide 2.5 [en línea] <https://www.optitrack.com/static/documents/NatNet%20API%20User%20Guide.pdf>

[8] V120:Trio Technical Drawings [en línea] <https://www.optitrack.com/static/documents/V120-Trio%20Technical%20Drawings.pdf>

[9] Cameras Pane [en línea] <http://wiki.optitrack.com/index.php?title=Cameras_Pane>

[10] Data Streaming Pane [en línea] <http://wiki.optitrack.com/index.php?title=Data_Streaming_Pane>

[11] Motive Streaming [en línea] <http://wiki.optitrack.com/index.php?title=Motive_Streaming>

[12] RFC3171 - IANA Guidelines for IPv4 Multicast Address Assignments [en línea] <http://tools.ietf.org/html/rfc3171>

[13] The Python Standard Library - Interpret strings as packet binary data [en línea] <https://docs.python.org/2/library/struct.html>

[14] PETER O. BASTA 2012. Quad Copter Flight. Master of Science in Electrical Engineering. Northidege, California State University.

[15] Elliott, Mark. "Stigmergic Collaboration: The Evolution of Group Work." M/C Journal 9.2 (2006). <http://journal.media-culture.org.au/0605/03-elliott.php>.

[16] Designing Collective Behavior in a Termite-Inspired Robot Construction Team [on line] http://www.uvm.edu/~cmplxsys/wp-content/uploads/werfel-termites-science-2014.pdf

[17] Leslie Marsh et al. Stigmergic epistemology, stigmeric cognition. MPRA Munich Personal RePEc Archive. Centre for Research in Cognitive Science, University of Sussex, United Kingdom, (2007).

[18] Inkyu Sa et al. Estimation and Control for an Open-Source Quadcopter. Proceedings of Australian Conference on Robotics and Automation, Monash University, Melbourne Australia. 2011.

[19] Samir Bouabdallah et al. Design and Control of a Miniature Quadrotor. Autonomous Systems Lab, ETH Zurich, Switzerland.

[20] Samir Bouabdallah 2007. Design and Control of Quadtrotors with Application to Autonomous flying. Grado de Doctor en Ciencias. École Polytechnique Fédérale de Lausanne EPFL.

ANEXOS

Código principal del sistema.

```
# -*- coding: cp1252 -*-

"""
Sistema CRAZYFLIE_OPTITRACK 2015_01
"""
## Motive:Tracker 120FPS - 50EXP - 200THR - 15LED   Preset Tracking

from datetime import datetime
import socket
import math
import time, sys
from threading import Thread
import struct
from struct import *

#FIXME: Has to be launched from within the example folder
sys.path.append("../lib")
import cflib
from cflib.crazyflie import Crazyflie

import logging
logging.basicConfig(level=logging.ERROR) #logging.basicConfig(level=logging.DEBUG)

#f=open('optiOutput.txt', 'w')
#f.write(time.asctime())
#f.write("\n\n")

xrb=[0.0,0.0,0.0,0.0]; xrbi=[0.0,0.0];    yrb=[0.0,0.0,0.0,0.0]; yrbi=[0.0,0.0];    zrb=[0.0,0.0,0.0,0.0];
zrbi=[0.0,0.0];
materialx=[0.0,0.0,0.0,0.0,0.0]; materialy=[0.0,0.0,0.0,0.0,0.0];
x00=[0.0]; y00=[0.0]; z00=[0.0];
x01=[0.0]; y01=[0.0]; z01=[0.0];
pos=[0.0,0.0,0.0,0.0,0.0];
colmenax=[0.0]; colmenay=[0.0];
startx=[0.0,0.0]; starty=[0.0,0.0];
espera=[False,False];
orirb=[0.0,0.0,0.0];
apagadoEmergencia=False;
frameNum=[0.0,0.0];
mError=[-1.0,-1.0,-1.0];
valores= ["\n"+"Valores del Quadcopter
ch10"+"\n"+"Num\tx\ty\tz\terror\t\troll\tpitch\tthrust\t\tx0\ty0\tz0"+"\n","\n"+"Valores del Quadcopter
ch80"+"\n"+"Num\tx\ty\tz\terror\t\troll\tpitch\tthrust\t\tx0\ty0\tz0"+"\n"];
horatomada=[" "];

class MotorRampExample:
```

```
    def __init__(self, link_uri):
        self._cf = Crazyflie()

        self._cf.connected.add_callback(self._connected)
        self._cf.disconnected.add_callback(self._disconnected)
        self._cf.connection_failed.add_callback(self._connection_failed)
        self._cf.connection_lost.add_callback(self._connection_lost)

        self._cf.open_link(link_uri)
        #print "Conectando a %s" % link_uri

    def _connected(self, link_uri):
        print "Conectado a ",link_uri
        if link_uri=="radio://0/10/250K":
            Thread(target=self._ramp_motors1).start()
        if link_uri=="radio://1/50/1M":
            Thread(target=self._ramp_motors2).start()
        if link_uri=="radio://1/80/2M":
            Thread(target=self._ramp_motors3).start()

    def _connection_failed(self, link_uri, msg):
        print "Connection to %s failed: %s" % (link_uri, msg)
        apagadoEmergencia=True;
    def _connection_lost(self, link_uri, msg):
        print "Connection to %s lost: %s" % (link_uri, msg)
        apagadoEmergencia=True;
    def _disconnected(self, link_uri):
        print "Desconectado de %s" % link_uri
        apagadoEmergencia=True;

    def _ramp_motors1(self): # con variacion por diferencia de distancias entre frames.
        contador=0;
        global valores; global x00; global y00; global z00; global xrbi; global yrbi; global espera;
        thrust0 = 40000; pitch0 = 0;  roll0 = 0;   yawrate0 = 0;    tiempoAntes=0;
        kpR=33.0;   #print "\tKP_Roll: ",kpR,"\tError de velocidad";                    #18.0   25.0    30.0
        kiR=0.8;    #print "\tKI_Roll: ",kiR,"\tError acumulado de velocidad: posicion";    # 0.5    0.7     1.0
        kdR=5.0;    #print "\tKD_Roll: ",kdR,"\tVariacion de Aceleracion";              # 2.0    5.0
        kpP=38.0;   #print "\tKP_Pitch: ",kpP,"\tError de velocidad";                   #18.0   25.0    30.0
        kiP=0.95;   #print "\tKI_Pitch: ",kiP,"\tError acumulado de velocidad: posicion";    # 0.5    0.7     1.0
        kdP=5.0;    #print "\tKD_Pitch: ",kdP,"\tVariacion de Aceleracion";              # 2.0    5.0
        kpT=500.0;  #print "\tKP_Thrust: ",kpT,"\tDiferencia de alturas";
        kiT=1.0;    #print "\tKI_Thrust: ",kiT,"\tCaida de voltaje";
        kdT=7500.0; #print "\tKD_Thrust: ",kdT,"\tVelocidad vertical\n";

        #print "roll0=   kpR*(((xrbi[0]-xAnt0)*100))   +  kiR*(int((xrbi[0]-x00)*100)) +   kdR*(((xrbi[0]-
2*xAnt0+xAntAnt0)*100));"
        #print "pitch0=  kpP*(((yAnt0-yrbi[0])*100))   +  kiP*(int((y00-yrbi[0])*100)) +   kdP*(((2*yAnt0-
yrbi[0]-yAntAnt0)*100));"
        #print "thrust0= kpT*(int((zrbi[0]-z00)*100))  +  kdT*(((zrbi[0]-zAnt0)*100))  +   kiT*zAcc0    +
38000;"; print "\n";

        tiempoInicio=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;

        xAcc0=0; yAcc0=0; zAcc0=0; alturaInicial=False;
        xAntAnt0=0; yAntAnt0=0;
```

```
        xrbi[0]=xrb[0]; xAnt0=xrbi[0]; x00=xrbi[0];   # posiciones iniciales
        yrbi[0]=yrb[0]; yAnt0=yrbi[0]; y00=yrbi[0];   # posiciones iniciales
        zrbi[0]=zrb[0]; zAnt0=zrbi[0]; z00=zrbi[0];   # posiciones iniciales
        #x00=-40.0/100; y00=10.0/100; z00=150.0/100;  # posicion target

        while thrust0 >= 0:#20000:
            contador=contador+1;
            if (apagadoEmergencia==True):   #Perdida de feedback o conexion con Optitrack
                break

            tiempoAhora=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;
            #print tiempoAhora-tiempoAntes, " milisegundos desde el frame anterior"
            #tiempoAntes=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;

            #self.rutina0(tiempoInicio, tiempoAhora);
            self.follow0();

            ### Control
##########################################################################################
##########
            xrbi[0]=xrb[0]; yrbi[0]=yrb[0]; zrbi[0]=zrb[0]; # valores del optitrack para este loop

            zAcc0 = zAcc0 + int((zrbi[0]-z00)*100); #Error Acumulado para termino integral de z

            roll0=   kpR*(((xrbi[0]-xAnt0)*100))   +  kiR*(int((xrbi[0]-x00)*100)) +   kdR*(((xrbi[0]-
2*xAnt0+xAntAnt0)*100));
            pitch0=  kpP*(((yAnt0-yrbi[0])*100))   +  kiP*(int((y00-yrbi[0])*100)) +   kdP*(((2*yAnt0-yrbi[0]-
yAntAnt0)*100));
            thrust0= kpT*(int((zrbi[0]-z00)*100))  +  kdT*(((zrbi[0]-zAnt0)*100))  +   kiT*zAcc0    + 38000;
            #thrust0=10100;
            if (espera[0]==True):
                thrust0=0;#10100;
                zAcc0=0;

            xAntAnt0=xAnt0; yAntAnt0=yAnt0;               # Posicion anterior anterior para siguiente loop
            xAnt0=xrbi[0]; yAnt0=yrbi[0]; zAnt0=zrbi[0];    # Posicion anterior para siguiente loop

            if (thrust0 >= 55000):   thrust0= 55000;   # limites de seguridad
            elif (thrust0 <= 0):     thrust0= 0;
            #if (zAcc0 >= 1000):      zAcc0=   1000; #elif (zAcc0 <= -1000):   zAcc0=   -1000;
            if (roll0 >= 15):        roll0 =  15;
            elif (roll0 <= -15):     roll0 =  -15;
            if (pitch0 >= 15):       pitch0 = 15;
            elif (pitch0 <= -15):    pitch0 = -15;

##########################################################################################
####################

            #self.choque();

            self._cf.commander.send_setpoint(roll0, pitch0, yawrate0, thrust0);
            valores[0]= valores[0] + str(contador)+"\t"+
str(int(xrbi[0]*100))+"\t"+str(int(yrbi[0]*100))+"\t"+str(int(zrbi[0]*100))+"\t"+str(round(mError[0]*1000,2))
```

```
+"\t\t"+str(round(roll0,2))+"\t"+str(round(pitch0,2))+"\t"+str(int(thrust0))+"\t\t"+str(round(x00*100,2))+"\t"+
str(round(y00*100,2))+"\t"+str(round(z00*100,2))+"\n";
          time.sleep(0.01);  # 100 comandos por segundo. Comandos enviados cada 10ms.

       self._cf.commander.send_setpoint(0, 0, 0, 0);
       time.sleep(0.1);
       print "Tiempo ch10: ",(tiempoAhora - tiempoInicio)/(60*60*1000),((tiempoAhora - tiempoInicio)-
(tiempoAhora - tiempoInicio)/(60*60*1000)*60*60*1000)/(60*1000),((tiempoAhora - tiempoInicio) -
(tiempoAhora - tiempoInicio)/(60*60*1000)*60*60*1000 -((tiempoAhora - tiempoInicio)-(tiempoAhora -
tiempoInicio)/(60*60*1000)*60*60*1000)/(60*1000)*60*1000)/(1000);
       print "Fin de Rutina para  radio://0/10/250K."
       self._cf.close_link();
       print "Numero de comandos: ",int(contador-1);
       print "Tiempo: ",tiempoAhora-tiempoInicio, " milisegundos";
       print "Tasa de comandos: ", int(((contador-1)*1000)/(tiempoAhora-tiempoInicio)),"
comandos/segundo\n";


    def _ramp_motors2(self): # con variacion por diferencia de distancias entre frames.
       global valores;
       thrust1 = 40000; pitch1 = 0;  roll1 = 0;   yawrate1 = 0;    tiempoAntes=0;
       contador=0;
       kpR=40.0;   print "\tKP_Roll: ",kpR,"\tError de velocidad";                    #20.0   25.0    30.0    35.0
       kiR=1.0;    print "\tKI_Roll: ",kiR,"\tError acumulado de velocidad: posicion";     # 0.2    0.5     0.7     1.0
       kdR=7.0;    print "\tKD_Roll: ",kdR,"\tVariacion de Aceleracion";              # 2.0    5.0
       kpP=40.0;   print "\tKP_Pitch: ",kpP,"\tError de velocidad";                   #18.0   25.0    30.0
       kiP=1.2;    print "\tKI_Pitch: ",kiP,"\tError acumulado de velocidad: posicion";    # 0.5    0.7     1.0
       kdP=3.0;    print "\tKD_Pitch: ",kdP,"\tVariacion de Aceleracion";              # 2.0    5.0
       kpT=500.0;  print "\tKP_Thrust: ",kpT,"\tDiferencia de alturas";
       kiT=1.0;    print "\tKI_Thrust: ",kiT,"\tCaida de voltaje";
       kdT=7500.0; print "\tKD_Thrust: ",kdT,"\tVelocidad vertical\n";

       #self._cf.commander.send_setpoint(0, 0, 0, 41000);
       #time.sleep(0.75);
       #self._cf.commander.send_setpoint(0, 0, 0, 0);
       tiempoInicio=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;

       xAcc1=0; yAcc1=0; zAcc1=0; alturaInicial=False;
       xAntAnt1=0; yAntAnt1=0;
       xrbi[1]=xrb[1]; xAnt1=xrbi[1]; x01=xrbi[1];   # posiciones iniciales
       yrbi[1]=yrb[1]; yAnt1=yrbi[1]; y01=yrbi[1];   # posiciones iniciales
       zrbi[1]=zrb[1]; zAnt1=zrbi[1];                # posiciones iniciales
       x01=0.0/100; y01=0.0/100; z01=160.0/100;              # posicion target

       while thrust1 >= 0:#20000:

          contador = contador+1;
          if (apagadoEmergencia==True):   #Perdida de feedback o conexion con Optitrack
             break

          tiempoAhora=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;
          #print tiempoAhora-tiempoAntes, " milisegundos desde el frame anterior"
          #tiempoAntes=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;
```

```
        ### Control
##########################################################################################
##########
        xrbi[1]=xrb[1]; yrbi[1]=yrb[1]; zrbi[1]=zrb[1]; # valores del optitrack para este loop

        zAcc1 = zAcc1 + int((zrbi[1]-z01)*100); #Error Acumulado para termino integral de z

        roll1=  kpR*(((xrbi[1]-xAnt1)*100))  +  kiR*(int((xrbi[1]-x01)*100)) +  kdR*(((xrbi[1]-
2*xAnt1+xAntAnt1)*100));
        pitch1=  kpP*(((yAnt1-yrbi[1])*100))  +  kiP*(int((y01-yrbi[1])*100)) +  kdP*(((2*yAnt1-yrbi[1]-
yAntAnt1)*100));
        thrust1= kpT*(int((zrbi[1]-z01)*100))  +  kdT*(((zrbi[1]-zAnt1)*100))  +  kiT*zAcc1   + 38000;
        #thrust1=10100;

        xAntAnt1=xAnt1; yAntAnt1=yAnt1;              # Posicion anterior anterior para siguiente loop
        xAnt1=xrbi[1]; yAnt1=yrbi[1]; zAnt1=zrbi[1];  # Posicion anterior para siguiente loop

        if (thrust1 >= 55000):  thrust1= 55000;  # limites de seguridad
        elif (thrust1 <= 0):    thrust1= 0;
        #if (zAcc1 >= 1000):     zAcc1=  1000; #elif (zAcc1 <= -1000):  zAcc1=  -1000;
        if (roll1 >= 20):       roll1 =  20;
        elif (roll1 <= -20):    roll1 =  -20;
        if (pitch1 >= 20):      pitch1 = 20;
        elif (pitch1 <= -20):   pitch1 = -20;

##########################################################################################
####################

        self._cf.commander.send_setpoint(roll1, pitch1, yawrate1, thrust1);
        valores[1]= valores[1] + str(contador)+"\t"+
str(int(xrbi[1]*100))+"\t"+str(int(yrbi[1]*100))+"\t"+str(int(zrbi[1]*100))+"\t"+str(round(mError[1]*1000,2))
+"\t\t"+str(round(roll1,2))+"\t"+str(round(pitch1,2))+"\t"+str(round(thrust1,2))+"\n";
        time.sleep(0.01);  # 100 comandos por segundo. Comandos enviados cada 10ms.

      self._cf.commander.send_setpoint(0, 0, 0, 0);
      time.sleep(0.2);
      print "Tiempo ch50: ",(tiempoAhora - tiempoInicio)/(60*60*1000),((tiempoAhora - tiempoInicio)-
(tiempoAhora - tiempoInicio)/(60*60*1000)*60*60*1000)/(60*1000),((tiempoAhora - tiempoInicio) -
(tiempoAhora - tiempoInicio)/(60*60*1000)*60*60*1000 -((tiempoAhora - tiempoInicio)-(tiempoAhora -
tiempoInicio)/(60*60*1000)*60*60*1000)/(60*1000)*60*1000)/(1000);
      print "Fin de Rutina para  radio://0/50/250K."
      self._cf.close_link();

   def _ramp_motors3(self): # con variacion por diferencia de distancias entre frames.
      contador=0;
      global valores; global x01; global y01; global z01; global xrbi; global yrbi; global espera;
      thrust1 = 0; pitch1 = 0;  roll1 = 0;  yawrate1 = 0;   tiempoAntes=0;
      kpR=33.0;  #print "\tKP_Roll: ",kpR,"\tError de velocidad";                    #18.0  25.0   30.0
      kiR=0.8;   #print "\tKI_Roll: ",kiR,"\tError acumulado de velocidad: posicion";   # 0.5   0.7    1.0
      kdR=5.0;   #print "\tKD_Roll: ",kdR,"\tVariacion de Aceleracion";              # 2.0   5.0
      kpP=35.0;  #print "\tKP_Pitch: ",kpP,"\tError de velocidad";                   #18.0  25.0   30.0
      kiP=0.7;   #print "\tKI_Pitch: ",kiP,"\tError acumulado de velocidad: posicion";   # 0.5   0.7    1.0
      kdP=5.0;   #print "\tKD_Pitch: ",kdP,"\tVariacion de Aceleracion";              # 2.0   5.0
      kpT=500.0;  #print "\tKP_Thrust: ",kpT,"\tDiferencia de alturas";
      kiT=1.0;   #print "\tKI_Thrust: ",kiT,"\tCaida de voltaje";
```

```
        kdT=7500.0; #print "\tKD_Thrust: ",kdT,"\tVelocidad vertical\n";

        tiempoInicio=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;

        xAcc1=0; yAcc1=0; zAcc1=0; alturaInicial=False;
        xAntAnt1=0; yAntAnt1=0;
        xrbi[1]=xrb[1]; xAnt1=xrbi[1]; x01=xrbi[1];   # posiciones iniciales
        yrbi[1]=yrb[1]; yAnt1=yrbi[1]; y01=yrbi[1];   # posiciones iniciales
        zrbi[1]=zrb[1]; zAnt1=zrbi[1]; z01=zrbi[1];   # posiciones iniciales
        #x01=40.0/100; y01=10.0/100; z01=150.0/100;   # posicion target

        while thrust1 >= 0:
            contador=contador+1;
            if (apagadoEmergencia==True):   #Perdida de feedback o conexion con Optitrack
                break

            tiempoAhora=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;
            #print tiempoAhora-tiempoAntes, " milisegundos desde el frame anterior"
            #tiempoAntes=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000;

            #self.rutina1(tiempoInicio, tiempoAhora);
            self.follow1();

            ### Control
##########################################################################################
##########
            xrbi[1]=xrb[1]; yrbi[1]=yrb[1]; zrbi[1]=zrb[1]; # valores del optitrack para este loop

            zAcc1 = zAcc1 + int((zrbi[1]-z01)*100); #Error Acumulado para termino integral de z

            roll1=   kpR*(((xrbi[1]-xAnt1)*100))   +  kiR*(int((xrbi[1]-x01)*100)) +   kdR*(((xrbi[1]-
2*xAnt1+xAntAnt1)*100));
            pitch1=  kpP*(((yAnt1-yrbi[1])*100))   +  kiP*(int((y01-yrbi[1])*100)) +   kdP*(((2*yAnt1-yrbi[1]-
yAntAnt1)*100));
            thrust1= kpT*(int((zrbi[1]-z01)*100))  +  kdT*(((zrbi[1]-zAnt1)*100))  +   kiT*zAcc1    + 38000;
            thrust1=10100;
            if (espera[1]==True):
                thrust1=0;
                zAcc1=0;

            xAntAnt1=xAnt1; yAntAnt1=yAnt1;                # Posicion anterior anterior para siguiente loop
            xAnt1=xrbi[1]; yAnt1=yrbi[1]; zAnt1=zrbi[1];    # Posicion anterior para siguiente loop

            if (thrust1 >= 55000):   thrust1= 55000;   # limites de seguridad
            elif (thrust1 <= 0):     thrust1= 0;
            #if (zAcc1 >= 1000):      zAcc1=   1000; #elif (zAcc1 <= -1000):   zAcc1=   -1000;
            if (roll1 >= 15):        roll1 =  15;
            elif (roll1 <= -15):     roll1 =  -15;
            if (pitch1 >= 15):       pitch1 = 15;
            elif (pitch1 <= -15):    pitch1 = -15;

##########################################################################################
####################
```

```
            #self.choque();

            self._cf.commander.send_setpoint(roll1, pitch1, yawrate1, thrust1);
            valores[1]= valores[1] + str(contador)+"\t"+
str(int(xrbi[1]*100))+"\t"+str(int(yrbi[1]*100))+"\t"+str(int(zrbi[1]*100))+"\t"+str(round(mError[1]*1000,2))
+"\t\t"+str(round(roll1,2))+"\t"+str(round(pitch1,2))+"\t"+str(int(thrust1))+"\t\t"+str(round(x01*100,2))+"\t"+
str(round(y01*100,2))+"\t"+str(round(z01*100,2))+"\n";
            #time.sleep(0.0085); # 120 comandos por segundo; Pierde la comunicacion rapidamente.
            time.sleep(0.01);  # 100 comandos por segundo. Comandos enviados cada 10ms.
            #time.sleep(0.02); # 50 comandos por segundo; Pierde la comunicacion rapidamente.
            #time.sleep(0.05); # 20 comandos por segundo; Pierde la comunicacion rapidamente.


        self._cf.commander.send_setpoint(0, 0, 0, 0);
        time.sleep(0.3);
        print "Tiempo ch80: ",(tiempoAhora - tiempoInicio)/(60*60*1000),((tiempoAhora - tiempoInicio)-
(tiempoAhora - tiempoInicio)/(60*60*1000)*60*60*1000)/(60*1000),((tiempoAhora - tiempoInicio) -
(tiempoAhora - tiempoInicio)/(60*60*1000)*60*60*1000 -((tiempoAhora - tiempoInicio)-(tiempoAhora -
tiempoInicio)/(60*60*1000)*60*60*1000)/(60*1000)*60*1000)/(1000);
        print "Fin de Rutina para  radio://1/80/2M."
        self._cf.close_link();
        print "Numero de comandos: ",int(contador-1);
        print "Tiempo: ",tiempoAhora-tiempoInicio, " milisegundos";
        print "Tasa de comandos: ", int(((contador-1)*1000)/(tiempoAhora-tiempoInicio)),"
comandos/segundo\n";


    def follow0(self): # quad10 sigue en XYZ a RigidBody3
        global x00; global y00; global z00; global espera;

        if (zrb[2]<1.73): # RB2 en mesa: zrb[2]=1.78
            espera[0]=False;
            x00= xrb[2];
            if  (yrb[2] > yrbi[0]): y00=yrb[2]-0.4;
            elif(yrb[2] < yrbi[0]): y00=yrb[2]+0.4;
            z00= zrb[2];

        elif (zrb[2]>=1.73):
            if (z00<1.90):
                z00=z00+0.002;
            if (zrbi[0]>=1.8 and z00>=1.80):
                espera[0]=True;

        if (x00<-0.5): x00=-0.5;
        elif (x00> 0.5): x00= 0.5;
        if (y00<-0.3): y00=-0.3;
        elif (y00> 0.3): y00= 0.3;
        if (z00< 1.30): z00= 1.30;

    def follow1(self): # quad80 sigue en XYZ a RigidBody4
        global x01; global y01; global z01; global espera;

        if (zrb[3]<1.73): # RB2 en mesa: zrb[3]=1.78
            espera[1]=False;
            x01= xrb[3];
```

```
        if  (yrb[3] > yrbi[1]): y01=yrb[3]-0.4;
        elif(yrb[3] < yrbi[1]): y01=yrb[3]+0.4;
        z01= zrb[3];

    elif (zrb[3]>=1.73):
        if (z01<1.90):
            z01=z01+0.002;
        if (zrbi[1]>=1.8 and z01>=1.80):
            espera[1]=True;

    if (x01<-0.5): x01=-0.5;
    elif (x01> 0.5): x01= 0.5;
    if (y01<-0.3): y01=-0.3;
    elif (y01> 0.3): y01= 0.3;
    if (z01< 1.30): z01= 1.30;


  def rutina0(self, inicio, ahora): # quad10 detecta y transporta material de forma autonoma a lugar detectable
      global x00; global y00; global z00; global ax0; global ay0; global colmenax; global colmenay; global
espera;

    if (ahora-inicio >= 0 and ahora-inicio < 100):          # Captura de Datos Iniciales          0   - 1
        colmenax[0]=xrb[2];         # posicion x de la colmena
        colmenay[0]=yrb[2];         # posicion y de la colmena
        ax0= materialx[0];          # posicion x del material
        ay0= materialy[0];          # posicion y del material
        #if (pos[0]==0): print "\tax0: ",ax0," ay0: ",ay0, ahora-inicio;
        #horatomada[0]= horatomada[0] + str(ahora-inicio)+" ";
        if (0<ax0): pos[0]=1;       # posicion relativa del material al centro de referencia
        if (0>ax0): pos[0]=2;
        startx[0]=x00;              # posicion x del quadcopter
        starty[0]=y00;              # posicion y del quadcopter
        espera[0]=True;
    if (ahora-inicio < 1000 and pos[0]==0):
        ax0= materialx[0];          # posicion x del material
        ay0= materialy[0];          # posicion y del material
        #if (pos[0]==0): print "\tax0: ",ax0," ay0: ",ay0, ahora-inicio;
        if (0<ax0): pos[0]=1;       # posicion relativa del material al centro de referencia
        if (0>ax0): pos[0]=2;

    if (ahora-inicio >= 1000 and ahora-inicio < 3000):      # Ir a punto de referencia            1   - 3
        #print horatomada[0];
        x00=0; y00=0; z00=160.0/100; espera[0]=False;
    if (ahora-inicio >= 3000 and ahora-inicio < 10000):     # Busca de material                   3   - 10
        if ((x00 < (ax0+0.1)) and pos[0]==1): x00=x00+0.001;
        if ((x00 > (ax0-0.1)) and pos[0]==2): x00=x00-0.001;
        y00=ay0;
        if (z00 < 1.65):   z00=z00+0.0015; #1.69
    if (ahora-inicio >= 10000 and ahora-inicio < 18000):    # Agarre de material                  10  - 13
        z00=150.0/100;
        if (ahora-inicio >= 13000):                     # Viaje a Colmena                   13  - 18
            x00=colmenax[0]; y00=colmenay[0];
            z00=150.0/100;

    if (ahora-inicio >= 18000 and ahora-inicio < 24000):    # Deposito de material en Colmena       18  - 24
considerar bajar mas rapido
```

```
        x00=colmenax[0]; y00=colmenay[0];
        if(z00<1.76):   z00=z00+0.003;
    if (ahora-inicio >= 24000 and ahora-inicio < 25000):    # Empuja material direccion correcta    24  - 25
        if (pos[0]==1):  x00=colmenax[0] - 0.1;
        if (pos[0]==2):  x00=colmenax[0] + 0.1;
        y00=colmenay[0];
        z00=160.0/100;

    #if (ahora-inicio >= 26000 and ahora-inicio < 24000):    # Soltar el contenedor vacio
    #    if(z00<1.76):   z00=z00+0.003;

    if (ahora-inicio >= 25000 and ahora-inicio < 30000):    # Regreso a punto de despegue          25  - 30
        if (y00 < starty[0]):  y00=y00+0.002;
        if (y00 > starty[0]):  y00=y00-0.002;
        if (x00 < startx[0]):  x00=x00+0.002;
        if (x00 > startx[0]):  x00=x00-0.002;
        z00=165.0/100;
    if (ahora-inicio >= 30000):                         # Aterrizaje                      30
        if (z00<=1.85):
            x00=x00+0.001; y00=y00+0.001; z00=z00+0.002;
        if (z00>=1.85):
            espera[0]=True;


  def rutina1(self, inicio, ahora): # quad80 detecta y transporta material de forma autonoma a lugar detectable
      global x01; global y01; global z01; global ax1; global ay1; global colmenax; global colmenay; global
espera;
      t=26000; #31000;

      if (ahora-inicio >= 0 and ahora-inicio < 100):            # Captura de Datos Iniciales
          colmenax[0]=xrb[2];
          colmenay[0]=yrb[2];
          ax1= materialx[1];
          ay1= materialy[1];
          if (0<ax1): pos[1]=1;
          if (0>ax1): pos[1]=2;
          startx[1]=x01;
          starty[1]=y01;

          if ((colmenay[0]-0.12 < materialy[1]) and (colmenay[0]+0.12 > materialy[1])):
              if((colmenax[0]-0.12 < materialx[1]) and (colmenax[0]+0.12 > materialx[1])):
                  ax1=materialx[0];
                  ay1=materialy[0];

      if (ahora-inicio >= 0 and ahora-inicio < 1000+t):         # Tiempo de espera
          espera[1]=True;

      if (ahora-inicio >= 1000+t and ahora-inicio < 5000+t):    # Ir a punto de referencia
          x01=0; y01=0; z01=160.0/100; espera[1]=False;
      if (ahora-inicio >= 5000+t and ahora-inicio < 12000+t):   # Busca de material
          if ((x01 < (ax1+0.1)) and pos[1]==1):  x01=x01+0.001;
          if ((x01 > (ax1-0.1)) and pos[1]==2):  x01=x01-0.001;
          y01=ay1;
          if (z01 < 1.69):   z01=z01+0.0015;
      if (ahora-inicio >= 12000+t and ahora-inicio < 20000+t):  # Agarre de material
          z01=150.0/100;
```

```
        if (ahora-inicio >= 15000+t):                    # Viaje a Colmena
            x01=colmenax[0]; y01=colmenay[0];
            z01=150.0/100;
    if (ahora-inicio >= 20000+t and ahora-inicio < 30000+t):    # Deposito de material en Colmena
        x01=colmenax[0]; y01=colmenay[0];
        if(z01<1.76):   z01=z01+0.003;
    if (ahora-inicio >= 30000+t and ahora-inicio < 32000+t):
        if (pos[1]==1):  x01=colmenax[0] - 0.1;
        if (pos[1]==2):  x01=colmenax[0] + 0.1;
        y01=colmenay[0];
        z01=160.0/100;
    if (ahora-inicio >= 32000+t and ahora-inicio < 35000+t):    # Regreso a punto de referencia
        x01=0; y01=0; z01=160.0/100;
    if (ahora-inicio >= 35000+t and ahora-inicio < 38000+t):    # Regreso a punto de despegue
        x01=startx[1]; y01=starty[1]; z01=165.0/100;
    if (ahora-inicio >= 38000+t):                    # Aterrizaje
        if (z01<=1.85):
            x01=x01+0.001; y01=y01+0.001; z01=z01+0.002;
        if (z01>=1.85):
            espera[1]=True;



def unPackData(major, minor,PacketIn):
    offset = 0;
    # message ID, nBytes
    messageID, nBytes = unpack(byteorder+'hh',PacketIn[offset:offset+4])
    offset += 4
    #print 'Message ID : ',messageID,'\nByte count : ',nBytes

    if (messageID == 7): # FRAME OF MOCAP DATA packet
        frameNumber,nMarkerSets = unpack(byteorder+'ii',PacketIn[offset:offset+8])
        frameNum[1]=frameNumber;
        offset += 8
        #print 'Frame # : ',frameNumber,'\nMarker Set Count : ', nMarkerSets
        #f.write("Frame # : "+str(frameNumber)+"\n")
        #f.write("Marker Set Count : "+str(nMarkerSets)+"\n")
        i=nMarkerSets
        while (i > 0):

            # marker name
            ns = PacketIn[offset:offset+255]
            szNamelen=ns.find('\0') #print ns, szNamelen
            szName = unpack(byteorder+str(szNamelen)+'s',PacketIn[offset:offset+szNamelen])[0]
            offset += szNamelen+1 # include the C zero char
            #print 'Model Name: ',szName
            #f.write("Model Name: "+szName+"\n")

            # marker data
            nMarkers = unpack(byteorder+'i',PacketIn[offset:offset+4])[0]
            offset += 4
            #print 'Marker Count : ',nMarkers
            #f.write("Model Count : "+str(nMarkers)+"\n")
            j=nMarkers
            while (j>0):
                x,y,z = unpack(byteorder+'fff',PacketIn[offset:offset+12])
```

```
                #print "\tx["+str(nMarkers-j+1)+"]="+str(x), " y["+str(nMarkers-j+1)+"]="+str(y), "
z["+str(nMarkers-j+1)+"]="+str(z)
                markersdata = "x["+str(nMarkers-j+1)+"]="+str(x)+"  y["+str(nMarkers-j+1)+"]="+str(y)+"
z["+str(nMarkers-j+1)+"]="+str(z)
                #f.write(str(markersdata)+"\n")

                offset += 12
                j=j-1
            i=i-1

        #unidentified markers
        nUMarkers = unpack(byteorder+'i',PacketIn[offset:offset+4])[0]
        offset += 4
        #print 'Unidentified Markercount=',nUMarkers
        i = nUMarkers
        while (i > 0):
            ux,uy,uz = unpack(byteorder+'fff',PacketIn[offset:offset+12])
            if (nUMarkers == i):
                materialx[0]=ux;
                materialy[0]=uy;
            if ((nUMarkers==3 and i==2) or (nUMarkers==2 and i==1)):
                materialx[1]=ux;
                materialy[1]=uy;
            if (nUMarkers==3 and i==1):
                materialx[2]=ux;
                materialy[2]=uy;
            offset += 12
            i=i-1

        #rigid bodies
        nRigidBodies = unpack(byteorder+'i',PacketIn[offset:offset+4])[0]
        offset += 4
        #print 'Rigid Body Count : ',nRigidBodies
        #f.write("Rigid Body Count : "+str(nRigidBodies)+"\n")
        nr = nRigidBodies
        while (nr > 0):
            rID = unpack(byteorder+'i',PacketIn[offset:offset+4])[0]
            offset += 4
            rx,ry,rz= unpack(byteorder+'fff',PacketIn[offset:offset+12])
            offset += 12
            rqx,rqy,rqz,rqw = unpack(byteorder+'ffff',PacketIn[offset:offset+16])
            offset += 16
            if(nRigidBodies == nr): # POSICION DE RIGIDBODY A
                xrb[0]=rx;
                yrb[0]=ry;
                zrb[0]=rz;
                #orirb[0]= math.atan2(2*(rqw*rqz-rqx*rqy),1-2*(rqz*rqz+rqx*rqx))*180/math.pi            #yaw
                #orirb[1]= math.asin(2*(rqw*rqx+rqy*rqz)/(rqw*rqw+rqx*rqx+rqy*rqy+rqz*rqz))*180/math.pi
#pitch
                #orirb[2]= math.atan2(2*(rqw*rqy-rqx*rqz),1-2*(rqy*rqy+rqx*rqx))*180/math.pi            #roll
            if((nRigidBodies==4 and nr==3)or(nRigidBodies==3 and nr==2)or(nRigidBodies==2 and nr==1)):
            #if(nRigidBodies == nr+1): # POSICION DE RIGIDBODY B
                xrb[1]=rx;
                yrb[1]=ry;
                zrb[1]=rz;
            if((nRigidBodies==4 and nr==2)or(nRigidBodies==3 and nr==1)):
```

```
            #if(nRigidBodies == nr+2): # POSICION DE RIGIDBODY C
               xrb[2]=rx;
               yrb[2]=ry;
               zrb[2]=rz;
            if(nRigidBodies==4 and nr==1):  # POSICION DE RIGIDBODY D
               xrb[3]=rx;
               yrb[3]=ry;
               zrb[3]=rz;
            #print "ID : ",rID;            f.write("ID : "+str(rID)+"\n")
            #print "pos: ",rx,ry,rz;       f.write("pos: "+str(rx,ry,rz)+"\n")
            #print "ori: ",rqx,rqy,rqz,rqw;f.write("ori: "+str(rqx,rqy,rqz,rqw)+"\n")

            ## associated marker positions
            nRigidMarkers = unpack(byteorder+'i',PacketIn[offset:offset+4])[0]
            offset += 4
            #if(commentsdata): print 'Marker count=',nRigidMarkers
            md = []
            markerID = []
            markersize = []
            for i in range(0,nRigidMarkers):
               md.extend(unpack(byteorder+'fff',PacketIn[offset:offset+12]))
               offset += 12
            if major >= 2:
               for i in range(0,nRigidMarkers):
                  markerID.append(unpack(byteorder+'i',PacketIn[offset:offset+4])[0])
                  offset += 4
               for i in range(0,nRigidMarkers):
                  markersize.append(unpack(byteorder+'f',PacketIn[offset:offset+4])[0])
                  offset += 4
            #    for i in range(0,nRigidMarkers):
            #       print 'Marker ',i+1,' ID=',markerID[i],' markerData=',md[3*i],md[3*i+1],md[3*i+2],'
markersize=',markersize[i]
            #else:
            #    for i in range(0,nRigidMarkers):
            #       print 'Marker ',i+1,' markerData=',md[3*i],md[3*i+1],md[3*i+2]

            # marker errors
            if major >= 2:
               markerError = unpack(byteorder+'f',PacketIn[offset:offset+4])[0]
               if(nRigidBodies == nr): mError[0]=markerError;
               if((nRigidBodies==4 and nr==3)or(nRigidBodies==3 and nr==2)or(nRigidBodies==2 and
nr==1)): mError[1]=markerError;
               offset += 4
               #print 'Mean marker error=',markerError
               #f.write("Mean marker error="+str(markerError)+"\n")


            nr = nr-1 # next rigid body

      #print "\n"
      #f.write("\n")


if __name__ == '__main__':

   print "Sistema CRAZYFLIE_OPTITRACK 2015_01_31\n"
```

```
    cflib.crtp.init_drivers(enable_debug_driver=False)
    print "Buscando Crazyflies..."
    available = cflib.crtp.scan_interfaces()
    print "Crazyflies encontrados: ",len(available)
    alturaInicial=False;

    for i in available:
        print " ",i[0]

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('', 1511))

    mreq = struct.pack("=4sl", socket.inet_aton("239.255.42.99"), socket.INADDR_ANY)
    sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
    sock.settimeout(1)#borrable si se requiere esperar por data

    if len(available) > 0:
        try:
            tiempoAntes1=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000
            data = sock.recv(2048)
            byteorder='@'#
            unPackData(2,0,data)#
            frameNum[0]=frameNum[1];
            print "\nConexion con Optitrack Data Streaming exitosa.\n"

            if available[0][0]=="radio://0/10/250K":    MotorRampExample("radio://0/10/250K");
            if available[0][0]=="radio://0/80/2M":      MotorRampExample("radio://1/80/2M");
            if (len(available)>1):
                if available[1][0]=="radio://0/10/250K":    MotorRampExample("radio://0/10/250K");
                if available[1][0]=="radio://0/80/2M":      MotorRampExample("radio://1/80/2M");

            #MotorRampExample("radio://0/10/250K"); time.sleep(0.1); if (len(available)>1):
MotorRampExample("radio://1/80/2M");

        except socket.error,e:
            print "\nSin conexion con Optitrack Data Streaming.\n"
            apagadoEmergencia=True

        if (apagadoEmergencia==False):
            cont=1.0;
            while True:
                try:
                    data1 = sock.recv(2048)
                    byteorder='@'
                    unPackData(2,0,data1)
                    cont=cont+1;
                except socket.error,e:
                    print "Conexion con Optitrack perdida.\n"
                    apagadoEmergencia=True
                    break
                except KeyboardInterrupt:
                    print "KeyboardInterrupt por Usuario.\n"
                    apagadoEmergencia=True
                    break
```

```
            #if (mError[0]==0 or mError[1]==0): print "UNTRACKEDs"; apagadoEmergencia=True; break;
        tiempoAhora1=(datetime.now()).hour*(60*60*1000)+ (datetime.now()).minute*60*1000+
(datetime.now()).second*1000+ (datetime.now()).microsecond/1000
        time.sleep(1);
        print "\nEstadisticas del Optitrack Data Streaming"
        print "Tiempo: ",tiempoAhora1-tiempoAntes1," milisegundos en total" #time.asctime()
        print "Numero de frames: ",int(cont-1);
        print "Tasa de captura: ",int((cont-1)/((tiempoAhora1-tiempoAntes1)/1000.0))," frames/segundo","\n";
        #print "Frames #:",frameNum[1]-frameNum[0],frameNum;

        print valores[0];
        print valores[1];
   else:
      print "No se han encontrado Crazyflies.\n"
   print "Finalizacion de programa."
```