UNIVERSITAT DE BARCELONA

Trabajo Final de Grado

GRADO EN INGENIERÍA INFORMÁTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Desarrollo de un Bot de Trading de Criptomonedas basado en Análisis Técnico

---

Autor: Marc Cabrera Marin

Profesor:	Eduardo Urruticoechea
Realizado en:	Facultat de Matemàtiques i Informàtica Universitat de Barcelona
Barcelona,	10 de junio de 2025

Abstract

This work details the development of a cryptocurrency trading bot based on technical analysis. The bot interacts with the Binance [1] and Bitfinex [2] exchanges, using the CCXT library for connectivity with Binance and Bitfinex's paper trading functionality for simulation with a graphical interface. Key features include a graphical user interface, log recording, transaction management, basic buy/sell operations, stop-loss and take-profit orders, tracking of global and per-strategy PnL, and candlestick charts with a double simple moving average (SMA). The bot mainly operates with the ETH/BTC and USDT/USD pairs, employing an SMA crossover strategy [3] to identify bullish or bearish trends and automate trading decisions, aiming to maximize profitability while managing risk.

The system architecture is composed of multiple specialized modules that ensure code modularity and maintainability. The main module (‘main.py‘) acts as the central controller and manages the modern user interface developed with Tkinter, while ‘conexion.py‘ handles communication with the different exchanges via REST APIs. Portfolio management (‘cartera.py‘) provides functions to monitor and manage assets, while the transaction system (‘transacciones.py‘) logs and processes all buy and sell operations.

The user interface features a modern and intuitive design organized into tabs, including a main dashboard with real-time status indicators, a trading section with advanced orders, a detailed portfolio with asset distribution charts, automated strategy management, and a full transaction history. The system incorporates visual status indicators showing exchange connectivity, available balance, selected trading pair, and the bot's overall status.

The strategy engine implements moving average crossover algorithms with configurable periods, enabling automatic detection of entry and exit signals. The stop-loss and take-profit functionalities provide automated risk management, while the PnL system (‘pnltracker.py‘) calculates and monitors profits and losses both globally and per individual strategy.

The charting capabilities (‘chart.py‘) use mplfinance to generate Japanese candlestick visualizations with overlaid technical indicators, enabling visual analysis of market trends and patterns. The system supports multiple timeframes from 1 minute to 1 day, adapting to different trading styles.

The implementation includes robust error-handling mechanisms and extensive logging to facilitate debugging and system monitoring. The bot is designed to operate autonomously with minimal supervision, executing predefined strategies and notifying the user of significant events.

The application represents a complete solution for traders looking to automate their cryptocurrency trading strategies, combining the power of technical analysis with an accessible interface and advanced risk management features.

Resumen

Este trabajo detalla el desarrollo de un bot de trading de criptomonedas basado en análisis técnico. El bot interactúa con los exchanges Binance [1] y Bitfinex [2], utilizando la biblioteca CCXT para la conectividad con Binance y la funcionalidad de paper trading de Bitfinex para simulación con interfaz gráfica. Las funcionalidades clave incluyen una interfaz gráfica de usuario, registro de logs, gestión de transacciones, operaciones básicas de compra/venta, órdenes stop-loss y take-profit, seguimiento del PnL global y por estrategia, y gráficos de velas con doble media móvil simple (SMA). El bot opera principalmente con los pares ETH/BTC y USDT/USD, empleando una estrategia de cruce de SMAs [3] para identificar tendencias alcistas o bajistas y automatizar las decisiones de trading, buscando maximizar la rentabilidad gestionando el riesgo.

La arquitectura del sistema está compuesta por múltiples módulos especializados que garantizan la modularidad y mantenibilidad del código. El módulo principal (‘main.py‘) actúa como controlador central y gestiona la interfaz de usuario moderna desarrollada con Tkinter, mientras que ‘conexion.py‘ se encarga de la comunicación con los distintos exchanges mediante APIs REST. La gestión de la cartera (‘cartera.py‘) proporciona funciones para monitorizar y administrar los activos, mientras que el sistema de transacciones (‘transacciones.py‘) registra y procesa todas las operaciones de compra y venta.

La interfaz de usuario presenta un diseño moderno e intuitivo organizado en pestañas que incluye un panel principal con indicadores de estado en tiempo real, una sección de trading con órdenes avanzadas, un portafolio detallado con gráficos de distribución de activos, gestión de estrategias automatizadas y un historial completo de transacciones. El sistema incorpora indicadores visuales de estado que muestran la conectividad con los exchanges, el saldo disponible, el par seleccionado y el estado general del bot.

El motor de estrategias implementa algoritmos de cruce de medias móviles con períodos configurables, permitiendo la detección automática de señales de entrada y salida. La funcionalidad de stop-loss y take-profit proporciona gestión automatizada del riesgo, mientras que el sistema PnL (‘pnltracker.py‘) calcula y monitoriza los beneficios y pérdidas tanto a nivel global como por estrategia individual.

Las capacidades gráficas (‘chart.py‘) utilizan mplfinance para generar visualizaciones de velas japonesas con indicadores técnicos superpuestos, permitiendo el análisis visual de tendencias y patrones del mercado. El sistema soporta múltiples marcos temporales desde 1 minuto hasta 1 día, adaptándose a diferentes estilos de trading.

La implementación incorpora mecanismos robustos de gestión de errores y un sistema de logging extensivo para facilitar la depuración y monitorización del sistema. El bot está diseñado para operar de manera autónoma con supervisión mínima, ejecutando estrategias predefinidas y notificando al usuario sobre eventos importantes.

La aplicación representa una solución completa para traders que buscan auto-

matizar sus estrategias de trading en criptomonedas, combinando la potencia del análisis técnico con una interfaz accesible y funcionalidades avanzadas de gestión de riesgos.

Resum

Aquest treball detalla el desenvolupament d'un bot de trading de criptomonedes basat en anàlisi tècnic. El bot interactua amb els exchanges Binance [1] i Bitfinex [2], utilitzant la biblioteca CCXT per la connectivitat amb Binance i la funcionalitat de paper trading de Bitfinex per simulació amb interfície gràfica. Les funcionalitats clau inclouen una interfície gràfica d'usuari, registre de logs, gestió de transaccions, operacions bàsiques de compra/venda, ordres stop-loss i take-profit, seguiment del PnL global i per estratègia, i gràfics de veles amb doble mitjana mòbil simple (SMA). El bot opera principalment amb els parells ETH/BTC i USDT/USD, emprant una estratègia de creua de SMAs [3] per identificar tendències alcistes o baixistes i automatitzar les decisions de trading, buscant maximitzar la rendibilitat gestionant el risc.

L'arquitectura del sistema està composta per múltiples mòduls especialitzats que garanteixen la modularitat i mantenibilitat del codi. El mòdul principal ('main.py') actua com a controlador central i gestiona la interfície d'usuari moderna desenvolupada amb Tkinter, mentre que 'conexion.py' s'encarrega de la comunicació amb els diferents exchanges mitjançant APIs REST. La gestió de la cartera ('cartera.py') proporciona funcions per monitoritzar i administrar els actius, mentre que el sistema de transaccions ('transacciones.py') registra i processa totes les operacions de compra i venda.

La interfície d'usuari presenta un disseny modern i intuïtiu organitzat en pestanyes que inclouen un dashboard principal amb indicadors d'estat en temps real, una secció de trading amb ordres avançades, un portafoli detallat amb gràfics de distribució d'actius, gestió d'estratègies automatitzades i un historial complet de transaccions. El sistema incorpora indicadors visuals d'estat que mostren la connectivitat amb els exchanges, el saldo disponible, el parell seleccionat i l'estat general del bot.

El motor d'estratègies implementa algoritmes de creuament de mitjanes mòbils amb períodes configurables, permetent la detecció automàtica de senyals d'entrada i sortida. La funcionalitat de stop-loss i take-profit proporciona gestió automatitzada del risc, mentre que el sistema PnL ('pnltracker.py') calcula i monitoritza els beneficis i pèrdues tant a nivell global com per estratègia individual.

Les capacitats gràfiques ('chart.py') utilitzen mplfinance per generar visualitzacions de veles japoneses amb indicadors tècnics superposats, permetent l'anàlisi visual de tendències i patrons de mercat. El sistema suporta múltiples timeframes des d'1 minut fins a 1 dia, adaptant-se a diferents estils de trading.

La implementació incorpora mecanismes robustos de gestió d'errors i logging extensiu per facilitar el debugging i monitorització del sistema. El bot està dissen-

yat per operar de manera autònoma amb supervisió mínima, executant estratègies predefinides i notificant l'usuari sobre esdeveniments importants.

L'aplicació representa una solució completa per traders que busquen automatitzar les seves estratègies de trading en criptomonedes, combinant la potència del anàlisi tècnic amb una interfície accessible i funcionalitats avançades de gestió de risc.

Agradecimientos

Agradezco sinceramente a mi profesor, Eduardo Urruticoechea, por su guía, su paciencia y su apoyo constante durante el desarrollo de este proyecto. Su experiencia y consejos han sido fundamentales para superar los desafíos encontrados. También quiero agradecer a mi familia y amigos por su paciencia y ánimo.

Índice general

Capítulo 1

Introducción

1.1. Motivación

La motivación principal detrás del desarrollo de este bot de trading de criptomonedas surge de una combinación de interés personal en los mercados financieros digitales y el reconocimiento de las limitaciones inherentes al trading manual. La naturaleza incesante y altamente volátil del mercado de criptomonedas exige una vigilancia constante y una toma de decisiones rápida y disciplinada, tareas que resultan agotadoras y propensas a errores para un operador humano.

La automatización, a través de un bot de trading, ofrece una solución atractiva a estos desafíos. Un bot puede monitorizar el mercado y ejecutar operaciones las 24 horas del día, sin fatiga ni distracciones. Además, al basar sus decisiones en reglas algorítmicas predefinidas, elimina la influencia de las emociones como el miedo o la codicia, que a menudo conducen a decisiones impulsivas y perjudiciales en el trading manual [4]. La capacidad de reaccionar a las señales del mercado de forma instantánea y sistemática representa una ventaja competitiva significativa.

El desarrollo de este proyecto también está impulsado por el deseo de aplicar conocimientos de ingeniería informática, análisis de datos y finanzas cuantitativas en un campo práctico y en constante evolución. La creación de un bot funcional implica abordar retos técnicos interesantes, como la integración con APIs de exchanges, el procesamiento de datos de mercado en tiempo real, la implementación de algoritmos de análisis técnico y el diseño de una interfaz de usuario intuitiva.

1.2. Objetivos

El objetivo general de este Trabajo Final de Grado es diseñar, desarrollar e implementar un bot de trading de criptomonedas funcional, basado en estrategias de análisis técnico, capaz de operar de forma automatizada en los exchanges Binance y Bitfinex.

Para alcanzar este objetivo general, se definen los siguientes objetivos específicos:

- Investigar y comprender los fundamentos del trading de criptomonedas, analizando las características del mercado, los principales activos (ETH/BTC, USDT/USD), el funcionamiento de los exchanges (Binance, Bitfinex) y las herramientas de análisis técnico relevantes, con especial énfasis en las medias móviles simples (SMA).
- Diseñar la arquitectura del bot con una estructura modular que incluya componentes para la conexión con exchanges, el motor de estrategia, la gestión de órdenes, el cálculo de PnL, el registro de logs y la interfaz gráfica de usuario (GUI).
- Implementar la conectividad con las APIs de Binance (utilizando la biblioteca CCXT) y Bitfinex, permitiendo obtener datos de mercado en tiempo real y ejecutar órdenes de compra/venta.
- Desarrollar la estrategia de trading basada en el cruce de medias móviles simples (una corta y una larga) para generar señales de compra y venta basadas en la identificación de tendencias alcistas o bajistas.
- Implementar la gestión de órdenes avanzada con capacidad para colocar órdenes de mercado, así como órdenes stop-loss y take-profit para gestionar el riesgo y asegurar ganancias potenciales.
- Crear una interfaz gráfica de usuario (GUI) intuitiva que permita configurar el bot, monitorizar su funcionamiento, visualizar el estado de la cartera, las operaciones realizadas, el PnL y los gráficos de precios con las medias móviles.
- Implementar un sistema robusto para registrar eventos importantes, operaciones, errores y decisiones del bot, facilitando la depuración y el análisis posterior.
- Probar y validar el bot en el entorno de paper trading de Bitfinex y, potencialmente, con fondos reales limitados en Binance, para evaluar la funcionalidad, fiabilidad y rendimiento de la estrategia.

El alcance del proyecto se centra en la implementación de la estrategia de cruce de medias móviles simples para los pares ETH/BTC y USDT/USD en los exchanges Binance y Bitfinex. Se excluyen estrategias más complejas, otros pares de criptomonedas y exchanges adicionales. La interfaz gráfica se desarrollará utilizando Tkinter en Python [5].

Capítulo 2

Planificación

El desarrollo de este proyecto se ha abordado siguiendo una metodología iterativa, combinando fases de investigación, diseño, implementación y pruebas. Esta aproximación ha permitido adaptar el desarrollo a los descubrimientos y desafíos surgidos durante el proceso.

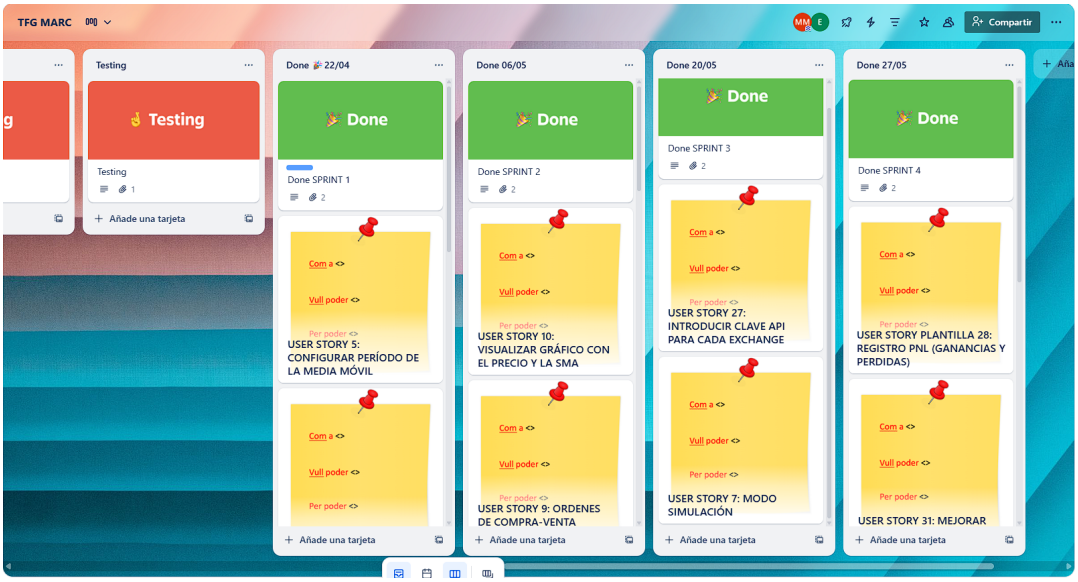

Figura 2.1: Planificación del proyecto en Trello con sprints y tareas

La planificación se organizó en sprints, como se muestra en la Figura 2.1, y se estructuró en las siguientes fases principales:

Los recursos clave utilizados en este proyecto incluyen Python 3.x como lenguaje de programación principal, las bibliotecas CCXT para interacción con exchanges, Tkinter para la GUI, Matplotlib y mplfinance para gráficos, y Pandas para manipulación de datos. Se utilizaron los exchanges Binance (para operaciones potenciales reales y obtención de datos) y Bitfinex (principalmente para paper trading y pruebas).

Fase	**Período**	**Actividades**
Investigación y Configuración	Semanas 1-2	Estudio del estado del arte, análisis de requisitos, selección de herramientas (Python, Tkinter, CCXT, Matplotlib), configuración del entorno y obtención de claves API
Desarrollo del Núcleo Lógico	Semanas 3-5	Implementación de conexiones con exchanges mediante CCXT, desarrollo del motor de estrategia SMA, lógica para obtención y procesamiento de datos de mercado
Desarrollo de la Interfaz Gráfica	Semanas 6-8	Diseño e implementación de GUI con Tkinter, visualización de datos de mercado, cartera, PnL, logs y gráficos de velas con indicadores
Integración y Gestión de Órdenes	Semanas 9-10	Implementación de órdenes (mercado, stop-loss, take-profit), integración de módulos y desarrollo del sistema de logging
Pruebas y Refinamiento	Semanas 11-13	Pruebas en paper trading de Bitfinex, validación de funcionalidad, robustez y rendimiento, depuración y ajustes de parámetros
Documentación	Semanas 14-16	Redacción de la memoria del TFG, descripción detallada del proyecto, metodología, resultados y conclusiones

Cuadro 2.1: Fases de desarrollo del proyecto

Capítulo 3

Contexto

3.1. El Mercado de Criptomonedas

El mercado de las criptomonedas ha experimentado un crecimiento exponencial en la última década, transformándose desde un nicho tecnológico hasta un ecosistema financiero global con una capitalización de mercado que alcanza billones de dólares. Activos digitales como Bitcoin (BTC) y Ethereum (ETH) se han consolidado como alternativas de inversión y medios de intercambio, atrayendo tanto a inversores institucionales como minoristas [6]. Sin embargo, este mercado se caracteriza por una volatilidad extrema y una operativa continua, 24 horas al día, 7 días a la semana, lo que presenta desafíos significativos para los traders humanos.

En este entorno dinámico, la automatización del trading mediante el uso de bots se ha convertido en una herramienta cada vez más relevante. Los bots de trading son programas informáticos diseñados para ejecutar estrategias de inversión de forma automática, basándose en algoritmos predefinidos y análisis de datos de mercado en tiempo real. Estos sistemas pueden operar de manera ininterrumpida, reaccionar instantáneamente a las fluctuaciones del mercado y eliminar el componente emocional inherente a las decisiones humanas, factores cruciales para tener éxito en el volátil mundo de las criptomonedas.

La evolución tecnológica ha facilitado el desarrollo y acceso a estos bots. Plataformas como Binance [1] y Bitfinex [2] no solo ofrecen una amplia gama de pares de criptomonedas para operar, sino que también proporcionan Interfaces de Programación de Aplicaciones (APIs) robustas que permiten a los desarrolladores conectar sus propios sistemas y automatizar operaciones. Bibliotecas como CCXT (CryptoCurrency eXchange Trading Library) [7] han estandarizado la interacción con múltiples exchanges, simplificando enormemente el proceso de desarrollo de bots compatibles con diversas plataformas.

Este proyecto se enmarca en este contexto, abordando la necesidad de herramientas automatizadas para navegar el complejo mercado de criptomonedas. El desarrollo de un bot de trading personalizado permite implementar estrategias específicas y adaptadas a las condiciones cambiantes del mercado, ofreciendo un potencial significativo para optimizar los resultados del trading.

3.1.1. Bots de Trading en Mercados Tradicionales

Es importante destacar que los bots de trading no son exclusivos del mercado de criptomonedas. De hecho, en los mercados financieros tradicionales (acciones, divisas, futuros, materias primas), los sistemas de trading automatizado han existido durante décadas, mucho antes del surgimiento de las criptomonedas. Desde los años 80, con la digitalización de las bolsas, los grandes bancos de inversión y fondos de cobertura comenzaron a implementar algoritmos para ejecutar operaciones de forma automática, evolucionando hasta los sofisticados sistemas de high-frequency trading (HFT) que dominan gran parte del volumen de operaciones en los mercados actuales.

Los bots de trading para mercados tradicionales y criptomonedas comparten principios fundamentales y estructuras similares: ambos analizan datos de mercado, aplican reglas predefinidas para tomar decisiones y ejecutan órdenes automáticamente. Las principales diferencias radican en las características específicas de cada mercado: los mercados tradicionales tienen horarios limitados, menor volatilidad, mayor regulación y liquidez, mientras que los mercados de criptomonedas operan 24/7, presentan mayor volatilidad y están menos regulados. Sin embargo, las estrategias técnicas como el cruce de medias móviles implementado en este proyecto son prácticamente idénticas en ambos entornos, lo que permite transferir conocimientos y técnicas entre ellos.

3.2. Criptomonedas: Fundamentos

Las criptomonedas son activos digitales diseñados para funcionar como medio de intercambio utilizando criptografía para asegurar las transacciones, controlar la creación de unidades adicionales y verificar la transferencia de activos. A diferencia de las monedas fiduciarias tradicionales, las criptomonedas operan en redes descentralizadas basadas en la tecnología blockchain, un libro mayor distribuido reforzado por una red de computadoras.

3.2.1. Características Principales

Las criptomonedas presentan varias características distintivas que las diferencian de los activos financieros tradicionales:

- **Descentralización:** Operan en redes distribuidas sin una autoridad central, lo que reduce la necesidad de confiar en intermediarios.
- **Transparencia:** Todas las transacciones se registran en la blockchain, que es pública y verificable por cualquier participante de la red.
- **Inmutabilidad:** Una vez confirmadas, las transacciones no pueden ser alteradas o revertidas, proporcionando seguridad y confianza.

- **Accesibilidad Global:** Permiten transacciones transfronterizas sin las restricciones y costos asociados a los sistemas financieros tradicionales.
- **Volatilidad:** Presentan fluctuaciones de precio significativas, lo que crea oportunidades de trading pero también aumenta el riesgo.
- **Oferta Limitada (en algunos casos):** Algunas criptomonedas, como Bitcoin, tienen una oferta máxima finita, lo que puede influir en su valor a largo plazo.

3.2.2. Bitcoin (BTC) y Ethereum (ETH)

Este proyecto se centra en dos de las criptomonedas más importantes:

- **Bitcoin (BTC):** La primera y más conocida criptomoneda, creada por Satoshi Nakamoto en 2009. Funciona como oro digital, una reserva de valor y un sistema de pago peer-to-peer.
- **Ethereum (ETH):** Una plataforma descentralizada que ejecuta contratos inteligentes: aplicaciones que se ejecutan exactamente como están programadas sin posibilidad de tiempo de inactividad, censura, fraude o interferencia de terceros. Ether (ETH) es la criptomoneda nativa de la plataforma Ethereum.

3.3. Exchanges de Criptomonedas

Los exchanges de criptomonedas son plataformas en línea que facilitan la compra, venta e intercambio de criptomonedas por otras criptomonedas o monedas fiduciarias. Actúan como intermediarios, conectando a compradores y vendedores.

3.3.1. Binance

Binance es uno de los exchanges de criptomonedas más grandes y populares del mundo por volumen de trading. Ofrece una amplia gama de servicios, incluyendo trading al contado, futuros, staking y más. Su API [1] es robusta y ampliamente utilizada por desarrolladores de bots.

3.3.2. Bitfinex

Bitfinex es otro exchange importante, conocido por sus herramientas avanzadas de trading y su liquidez. Ofrece una funcionalidad de "paper trading"que permite a los usuarios probar estrategias de trading con fondos simulados, lo cual es invaluable para el desarrollo y prueba de bots como el de este proyecto. Su API [2] también es completa.

3.4. APIs y Bibliotecas para Trading

Las Interfaces de Programación de Aplicaciones (APIs) son cruciales para el trading algorítmico, ya que permiten a los programas interactuar con los exchanges de forma automatizada.

3.4.1. APIs de Exchanges

Las APIs de Binance y Bitfinex proporcionan puntos de conexión para obtener datos de mercado en tiempo real (precios, volúmenes, libros de órdenes), gestionar cuentas (saldos, historial) y ejecutar órdenes de trading (compra, venta, cancelación).

3.4.2. Biblioteca CCXT

CCXT (CryptoCurrency eXchange Trading Library) [7] es una biblioteca de JavaScript / Python / PHP que proporciona una interfaz unificada para interactuar con más de 100 exchanges de criptomonedas. Simplifica enormemente el desarrollo de bots al abstraer las diferencias entre las APIs de los distintos exchanges, permitiendo escribir código que puede funcionar con múltiples plataformas con cambios mínimos.

3.5. Automatización del Trading

La automatización del trading implica el uso de programas informáticos (bots) para ejecutar estrategias de trading de forma automática. Los bots pueden analizar datos de mercado, tomar decisiones basadas en reglas predefinidas y ejecutar operaciones sin intervención humana.

3.5.1. Ventajas

- **Operación Continua:** Los bots pueden operar 24/7, aprovechando oportunidades en cualquier momento.
- **Velocidad de Reacción:** Pueden procesar información y ejecutar órdenes mucho más rápido que un humano.
- **Eliminación de Emociones:** Las decisiones se basan en lógica y algoritmos, eliminando el sesgo emocional.
- **Backtesting:** Permiten probar estrategias con datos históricos para evaluar su rendimiento potencial.
- **Disciplina:** Ejecutan la estrategia de forma consistente sin desviaciones.

3.5.2. Desventajas y Riesgos

- **Errores de Programación:** Un bug en el código puede llevar a pérdidas significativas.
- **Sobreoptimización (Overfitting):** Una estrategia que funciona bien en datos históricos puede no funcionar en el mercado real.
- **Fallos Técnicos:** Problemas de conexión, fallos del servidor del exchange o del propio bot.
- **Cambios en el Mercado:** Las condiciones del mercado pueden cambiar, haciendo que una estrategia previamente rentable deje de serlo.
- **Seguridad de las APIs:** Las claves API deben gestionarse de forma segura para evitar accesos no autorizados.

3.6. Análisis Técnico

El análisis técnico es una metodología para pronosticar la dirección de los precios mediante el estudio de datos pasados del mercado, principalmente el precio y el volumen. Se basa en la idea de que los movimientos históricos de precios y los patrones tienden a repetirse.

3.6.1. Indicadores Técnicos

Los indicadores técnicos son cálculos matemáticos basados en el precio, el volumen o el interés abierto de un valor o contrato. Los traders los utilizan para tomar decisiones de compra o venta.

Medias Móviles Simples (SMA)

Una Media Móvil Simple (SMA) [3] es un indicador de análisis técnico que calcula el precio medio de un activo durante un período específico. Se utiliza para suavizar las fluctuaciones de precios y ayudar a identificar la dirección de la tendencia.

Cruce de Medias Móviles

La estrategia de cruce de medias móviles es una técnica popular que utiliza dos SMAs con diferentes períodos (una corta y una larga). Se genera una señal de compra cuando la SMA corta cruza por encima de la SMA larga, y una señal de venta cuando la SMA corta cruza por debajo de la SMA larga.

Capítulo 4

Análisis

4.1. Estado del Arte

El desarrollo de bots de trading de criptomonedas es un campo activo y en constante evolución. A continuación, se analizan en profundidad algunas de las soluciones más relevantes del mercado, sus tecnologías y características distintivas:

4.1.1. Soluciones Comerciales

- **3Commas:** Plataforma basada en la nube que utiliza Node.js y React para su frontend. Ofrece estrategias como DCA (Dollar Cost Averaging), GRID y bots de arbitraje. Su arquitectura está diseñada para alta disponibilidad con balanceadores de carga y bases de datos distribuidas. Soporta más de 23 exchanges pero carece de personalización profunda en las estrategias y tiene costos de suscripción elevados (desde $29 hasta $99 mensuales) [**3commas˙docs**].
- **CryptoHopper:** Desarrollado en PHP y JavaScript, con una arquitectura orientada a servicios. Implementa estrategias basadas en indicadores técnicos como RSI, MACD y Bollinger Bands. Destaca por su marketplace de estrategias pero presenta limitaciones en la frecuencia de operaciones y personalización del código fuente. Su modelo de precios escala según el número de posiciones simultáneas (desde $19 hasta $99 mensuales) [**cryptohopper˙api**].
- **HaasOnline:** Utiliza C# y .NET para su backend, con una arquitectura cliente-servidor. Ofrece más de 50 indicadores técnicos y permite crear estrategias complejas mediante un editor visual. Su principal desventaja es la curva de aprendizaje pronunciada y el alto costo (desde 0.02 BTC por trimestre) [**haasonline˙tech**].

4.1.2. Soluciones de Código Abierto

- **Freqtrade:** Framework en Python que utiliza SQLAlchemy para persistencia de datos y FastAPI para su interfaz REST. Implementa backtesting, optimización de hiperparámetros y múltiples estrategias. Su arquitectura modular permite extensiones, pero requiere conocimientos avanzados de Python y presenta limitaciones en la interfaz gráfica [**freqtrade˙github**].
- **Gekko:** Desarrollado en JavaScript/Node.js con Express para el backend y Vue.js para el frontend. Utiliza una arquitectura de plugins para estrategias e indicadores. Ofrece backtesting y paper trading, pero su desarrollo se ha estancado desde 2019 y tiene limitaciones en el manejo de datos históricos [**gekko˙github**].
- **Zenbot:** Implementado en Node.js con MongoDB para almacenamiento. Destaca por su enfoque en estrategias de alta frecuencia y machine learning. Su principal desventaja es la documentación limitada y la complejidad de configuración [**zenbot˙docs**].

4.1.3. Comparativa y Justificación del Desarrollo Propio

La decisión de desarrollar un bot propio en lugar de utilizar soluciones existentes se basa en varios factores críticos:

Aspecto	**Soluciones Existentes**	**Bot Desarrollado**
Personalización	Limitada a parámetros predefinidos o requiere conocimientos avanzados	Control total sobre la lógica de trading y adaptación a necesidades específicas
Integración Multi-Exchange	Generalmente limitada a un subconjunto de funcionalidades por exchange	Implementación específica para Binance y Bitfinex con todas sus características particulares
Costos	Suscripciones mensuales o comisiones por operación	Costo único de desarrollo
Seguridad	Dependencia de servidores de terceros para claves API	Control total sobre la gestión de claves y datos sensibles
Enfoque Educativo	Caja negra con limitada comprensión interna	Comprensión profunda de cada componente y proceso
Interfaz de Usuario	Genérica, no optimizada para estrategia específica	Diseñada específicamente para la estrategia SMA con visualización optimizada

Cuadro 4.1: Comparativa entre soluciones existentes y bot desarrollado

Además de estos factores, el desarrollo propio permite:

- **Implementación de lógica específica:** La estrategia SMA implementada está optimizada para los pares ETH/BTC y USDT/USD con parámetros ajustados específicamente para su volatilidad y comportamiento.
- **Integración híbrida:** La combinación de Binance (mediante CCXT) para operaciones reales y Bitfinex para paper trading con interfaz gráfica no está disponible en ninguna solución comercial.
- **Control sobre la gestión de riesgos:** Implementación personalizada de stop-loss y take-profit adaptada a la estrategia específica y a la tolerancia al riesgo del usuario.
- **Valor académico:** El desarrollo desde cero proporciona un entendimiento profundo de los principios de trading algorítmico, análisis técnico y desarrollo de software financiero.

Esta comparativa demuestra que, aunque existen numerosas soluciones en el mercado, el desarrollo de un bot propio ofrece ventajas significativas en términos de control, personalización, seguridad y aprendizaje que justifican la inversión de tiempo y recursos en su implementación.

4.2. Público Objetivo

El bot de trading desarrollado en este proyecto está dirigido a:

- **Traders Minoristas:** Individuos con interés en el trading de criptomonedas que buscan automatizar sus estrategias y mejorar su eficiencia.
- **Desarrolladores y Estudiantes:** Personas con conocimientos de programación que desean aprender sobre el desarrollo de bots de trading y experimentar con estrategias algorítmicas.
- **Entusiastas de las Criptomonedas:** Usuarios que buscan herramientas para interactuar con el mercado de criptomonedas de una manera más sofisticada.

4.3. Requisitos Funcionales

Los requisitos funcionales definen las operaciones y funcionalidades que el sistema debe ser capaz de realizar:

4.3.1. Conexión y Datos

- RF1: El sistema debe permitir la conexión a los exchanges Binance y Bitfinex utilizando claves API proporcionadas por el usuario.

- RF2: El sistema debe ser capaz de obtener datos de mercado en tiempo real (precios, velas OHLCV) para los pares ETH/BTC y USDT/USD.
- RF3: El sistema debe permitir al usuario seleccionar el exchange, el par de trading y el intervalo de tiempo (timeframe) para el análisis.

4.3.2. Estrategia de Trading

- RF4: El sistema debe implementar la estrategia de trading basada en el cruce de dos medias móviles simples (SMA corta y SMA larga).
- RF5: El sistema debe permitir al usuario configurar los períodos de las SMAs.
- RF6: El sistema debe generar señales de compra cuando la SMA corta cruce por encima de la SMA larga.
- RF7: El sistema debe generar señales de venta cuando la SMA corta cruce por debajo de la SMA larga.

4.3.3. Gestión de Órdenes

- RF8: El sistema debe ser capaz de ejecutar órdenes de compra y venta de mercado basadas en las señales de la estrategia.
- RF9: El sistema debe permitir la configuración y ejecución automática de órdenes stop-loss para limitar pérdidas.
- RF10: El sistema debe permitir la configuración y ejecución automática de órdenes take-profit para asegurar ganancias.
- RF11: El sistema debe mostrar el estado de las órdenes activas y el historial de transacciones.

4.3.4. Interfaz Gráfica (GUI)

- RF12: El sistema debe proporcionar una interfaz gráfica intuitiva para la configuración, monitorización y operación del bot.
- RF13: La GUI debe mostrar gráficos de velas con las SMAs superpuestas y las señales de trading.
- RF14: La GUI debe mostrar el saldo de la cartera y el Profit and Loss (PnL) global y por estrategia.
- RF15: La GUI debe mostrar un registro (log) de eventos importantes, operaciones y errores.

4.4. Requisitos No Funcionales

Los requisitos no funcionales describen las cualidades y restricciones del sistema:

4.4.1. Rendimiento

- RNF1: El sistema debe procesar los datos de mercado y generar señales de trading con baja latencia.
- RNF2: La interfaz gráfica debe ser responsiva y actualizarse en tiempo real sin bloqueos.
- RNF3: El sistema debe ser capaz de manejar múltiples operaciones y conexiones concurrentes.

4.4.2. Seguridad

- RNF4: Las claves API deben almacenarse de forma segura y nunca exponerse en el código fuente.
- RNF5: Las comunicaciones con las APIs de los exchanges deben realizarse a través de conexiones seguras (HTTPS).
- RNF6: El sistema debe implementar mecanismos de validación para prevenir órdenes erróneas o potencialmente peligrosas.

4.4.3. Usabilidad

- RNF7: La interfaz gráfica debe ser intuitiva y permitir la configuración del bot sin conocimientos técnicos avanzados.
- RNF8: El sistema debe proporcionar feedback visual claro sobre el estado de las operaciones y conexiones.
- RNF9: Los gráficos y visualizaciones deben ser claros y fácilmente interpretables.

4.4.4. Fiabilidad

- RNF10: El sistema debe manejar adecuadamente errores de conexión, reintentos y recuperación.
- RNF11: El sistema debe implementar mecanismos de logging detallados para facilitar la depuración.
- RNF12: El sistema debe validar los datos recibidos de los exchanges antes de procesarlos.

4.4.5. Mantenibilidad

- RNF13: El código debe seguir principios de diseño modular y orientado a objetos.
- RNF14: El sistema debe estar estructurado para permitir la adición de nuevas estrategias o exchanges con cambios mínimos.
- RNF15: El código debe incluir comentarios y documentación adecuados.

4.5. Retos Técnicos

El desarrollo del bot de trading presenta varios desafíos técnicos significativos:

- **Integración con Múltiples Exchanges:** Cada exchange tiene sus propias peculiaridades en términos de API, formatos de datos y comportamientos. La biblioteca CCXT ayuda a estandarizar estas interacciones, pero aún requiere manejar casos específicos para cada exchange.
- **Gestión de la Concurrencia:** El bot debe realizar múltiples tareas simultáneamente: monitorizar el mercado, actualizar la interfaz gráfica, ejecutar órdenes y mantener conexiones activas. Esto requiere una gestión cuidadosa de la concurrencia para evitar bloqueos o condiciones de carrera.
- **Manejo de Errores y Reconexión:** Las conexiones con las APIs pueden fallar por diversas razones (problemas de red, mantenimiento del exchange, límites de tasa). El bot debe detectar estos fallos y reconectar de forma robusta.
- **Rendimiento de la Interfaz Gráfica:** La visualización de gráficos de velas con indicadores técnicos puede ser intensiva en recursos. Es necesario optimizar el rendimiento para mantener la interfaz responsiva incluso durante operaciones de trading activas.
- **Precisión en la Detección de Señales:** La detección precisa de cruces de medias móviles es crucial para la estrategia. Pequeños errores en el cálculo o en la temporización pueden llevar a señales falsas o perdidas.
- **Gestión del Riesgo:** Implementar mecanismos efectivos de stop-loss y take-profit requiere un manejo cuidadoso de las órdenes y una monitorización constante de las posiciones abiertas.
- **Seguridad:** La protección de las claves API y la validación de operaciones son críticas para prevenir errores costosos o accesos no autorizados.

Estos retos técnicos han guiado las decisiones de diseño e implementación descritas en los siguientes capítulos, buscando soluciones que equilibren funcionalidad, rendimiento, usabilidad y seguridad.

Capítulo 5

Diseño

Este capítulo describe las decisiones de diseño tomadas para el desarrollo del bot de trading, incluyendo la arquitectura general del sistema, el diseño de la interfaz gráfica, la estrategia de trading implementada y los mecanismos de gestión de órdenes y riesgos.

5.1. Decisiones de diseño

5.1.1. Arquitectura del Sistema

La arquitectura del sistema se ha diseñado siguiendo principios de modularidad y separación de responsabilidades, como se muestra en la siguiente figura:

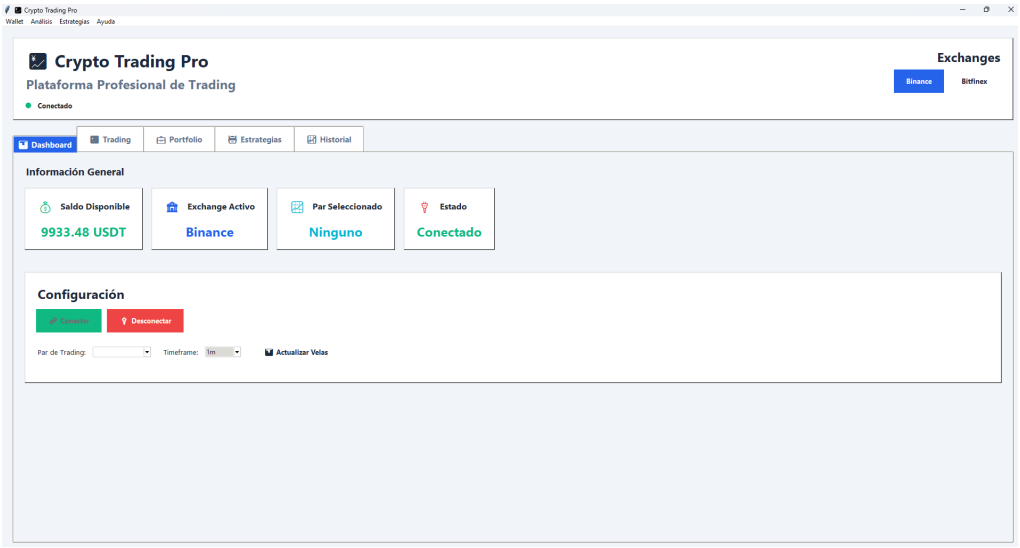

Figura 5.1: Interfaz principal del bot mostrando la arquitectura modular

5.1.2. Tecnologías y Patrones

- **Tecnologías principales:** Python 3.x, Tkinter, CCXT [7], Matplotlib
- **Patrones de diseño:** Singleton (conexiones), Observer (actualizaciones UI), Strategy (algoritmos trading)
- **Seguridad:** Gestión segura de credenciales API, validación de entradas, manejo defensivo de errores

5.1.3. Flujo de Operaciones

El siguiente diagrama muestra el flujo de operaciones del bot:

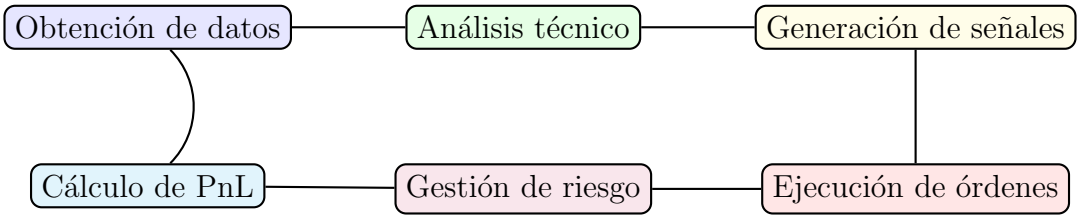

Figura 5.2: Flujo de operaciones del bot de trading

5.1.4. Visualización de Datos

La visualización de datos de mercado es fundamental para el análisis técnico:

Figura 5.3: Visualización de velas y SMAs para BTC/USDT

- **Limitación de Privilegios:** Las claves API utilizadas tienen solo los privilegios mínimos necesarios para la operación del bot.

5.2. Arquitectura del Sistema

La arquitectura general del bot de trading se ha diseñado siguiendo un enfoque modular y orientado a objetos, con componentes claramente definidos que interactúan entre sí para proporcionar la funcionalidad completa del sistema.

5.2.1. Componentes Principales

Los principales componentes del sistema son:

- **CryptoApp (main.py):** Componente principal que coordina todos los demás módulos y gestiona la interfaz gráfica de usuario. Actúa como el punto de entrada de la aplicación y orquesta la interacción entre los diferentes componentes.
- **ExchangeManager (conexion.py):** Gestiona las conexiones con los exchanges (Binance y Bitfinex) y proporciona una interfaz unificada para interactuar con ellos. Encapsula toda la lógica relacionada con la autenticación, obtención de datos de mercado y ejecución de órdenes.
- **ChartManager (chart.py):** Responsable de la obtención y procesamiento de datos de mercado, así como de la generación de gráficos de velas e indicadores. Utiliza Matplotlib y mplfinance para crear visualizaciones interactivas.
- **CarteraManager (cartera.py):** Gestiona la información sobre los saldos y posiciones del usuario en los diferentes exchanges. Mantiene un registro actualizado de los activos disponibles y su valor.
- **Transacciones (transacciones.py):** Maneja la ejecución de órdenes de compra, venta, stop-loss y take-profit. Implementa la lógica para validar, enviar y monitorizar órdenes en los exchanges.
- **PnLTracker (pnl_tracker.py):** Calcula y registra el rendimiento (Profit and Loss) de las operaciones realizadas. Proporciona métricas tanto globales como específicas de la estrategia.
- **Calculos (calculos.py):** Contiene los algoritmos para el cálculo de indicadores técnicos como medias móviles simples y RSI. Implementa la lógica matemática necesaria para la estrategia de trading.
- **Config (config.py):** Almacena constantes de configuración, credenciales de API y otros parámetros del sistema. Centraliza la configuración para facilitar su mantenimiento.

5.2.2. Flujo de Datos y Diagramas

5.3. Diseño de la Interfaz Gráfica

La interfaz gráfica de usuario (GUI) se ha diseñado priorizando la claridad, la usabilidad y la presentación efectiva de la información relevante para el trading. Se ha implementado utilizando Tkinter con estilos modernos definidos en el módulo ui_styles.py.

5.3.1. Principios de Diseño de la GUI

- **Simplicidad:** Interfaz limpia y no sobrecargada, centrada en la información esencial.
- **Organización Lógica:** Agrupación de elementos relacionados y flujo de trabajo intuitivo.
- **Feedback Visual:** Indicadores claros del estado del sistema.

5.4. Interfaz Gráfica de Usuario

La interfaz gráfica de usuario (GUI) del bot se ha diseñado con un enfoque en la usabilidad y la claridad, permitiendo al usuario monitorizar el mercado, configurar la estrategia y ejecutar operaciones de forma intuitiva.

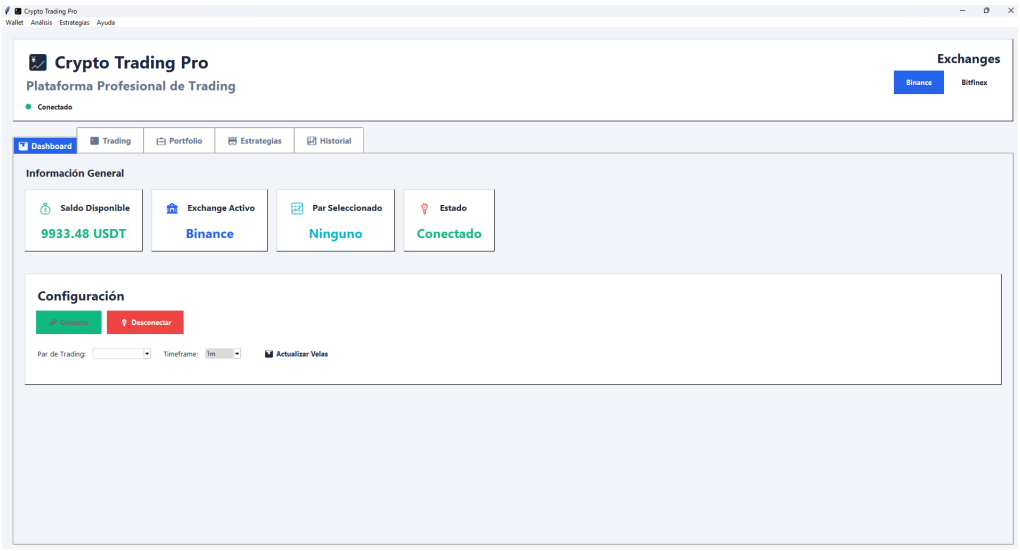

Figura 5.4: Página principal de la interfaz del bot de trading

5.4.1. Visualización del Portfolio

El sistema permite visualizar el portfolio completo del usuario, mostrando el saldo total y los activos disponibles:

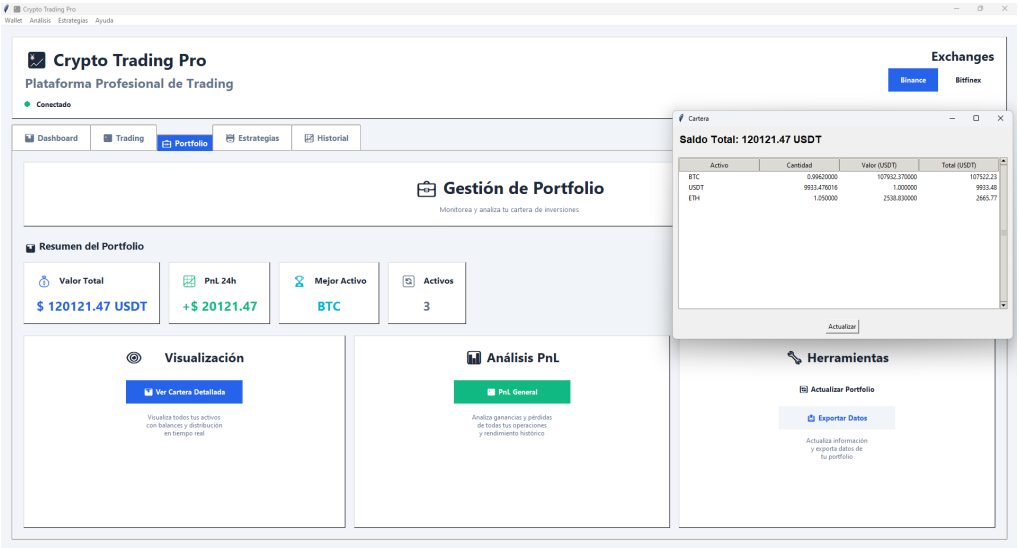

Figura 5.5: Visualización del portfolio y saldo total de la cartera con los activos en Binance

5.4.2. Componentes Personalizados

Se han implementado componentes modernos en el módulo ui_styles.py:

- **ModernButton:** Botones con efectos hover y estilos personalizados
- **ModernFrame:** Contenedores con bordes redondeados y sombras
- **StatusIndicator:** Indicadores visuales del estado de conexión
- **InfoCard:** Tarjetas informativas para mostrar métricas clave

5.5. Estrategia de Trading: Cruce de Medias Móviles

La estrategia de trading implementada en este bot se basa en el cruce de dos Medias Móviles Simples (SMA): una SMA corta y una SMA larga. Esta es una estrategia de seguimiento de tendencia clásica.

Figura 5.6: Gráfico BTC/USDT con línea de 5 minutos y SMA corto de 25 (rojo) y largo 50 (morado)

5.5.1. Lógica de la Estrategia

- **Señal de Compra (Alcista):** Se genera cuando la SMA de período corto cruza por encima de la SMA de período largo. Esto sugiere que el impulso a corto plazo es más fuerte que el impulso a largo plazo, indicando una posible tendencia alcista.
- **Señal de Venta (Bajista):** Se genera cuando la SMA de período corto cruza por debajo de la SMA de período largo. Esto sugiere que el impulso a corto plazo es más débil que el impulso a largo plazo, indicando una posible tendencia bajista.

5.5.2. Fórmulas y Cálculo de SMA

La Media Móvil Simple (SMA) es uno de los indicadores técnicos más fundamentales y ampliamente utilizados. Su cálculo matemático es relativamente sencillo pero poderoso:

$$SMA_n = \frac{P_1 + P_2 + P_3 + ... + P_n}{n} \tag{5.5.1}$$

Donde:

- SMA_n es la Media Móvil Simple para un período de n
- $P_1, P_2, ..., P_n$ son los precios de cierre para los últimos n períodos
- n es el número de períodos (días, horas, minutos, etc. dependiendo del timeframe)

En la implementación del bot, el cálculo se realiza mediante la siguiente función:

$$SMA_n(t) = \frac{\sum_{i=t-n+1}^{t} P_i}{n} \tag{5.5.2}$$

Donde t representa el índice temporal actual.

Para la detección de cruces, se comparan dos SMAs con diferentes períodos:

- **Señal de compra (alcista):** $SMA_{corto}(t) > SMA_{largo}(t)$ y $SMA_{corto}(t-1) \leq SMA_{largo}(t-1)$
- **Señal de venta (bajista):** $SMA_{corto}(t) < SMA_{largo}(t)$ y $SMA_{corto}(t-1) \geq SMA_{largo}(t-1)$

Estas condiciones se implementan en el código de la siguiente manera:

```
# Señal de compra alcista:
if (ultimo_sma_corto > ultimo_sma_largo and sma_corto[-2] <= sma_largo[-2]):
    # SMA corto cruza POR ENCIMA del SMA largo
    senal_actual = "Compra (Alcista)"

# Señal de venta bajista:
elif (ultimo_sma_corto < ultimo_sma_largo and sma_corto[-2] >= sma_largo[-2]):
    # SMA corto cruza POR DEBAJO del SMA largo
    senal_actual = "Venta (Bajista)"
```

Figura 5.7: Gráfico BTC/USDT con velas de 5 minutos y SMAs: corto de 25 períodos (rojo) y largo de 50 períodos (morado)

5.6. Gestión de Órdenes y Riesgos

Una gestión eficaz de las órdenes y del riesgo es fundamental para el éxito del trading algorítmico.

5.6.1. Tipos de Órdenes

El bot implementa los siguientes tipos de órdenes:

- **Órdenes de Mercado (Market Orders):** Se ejecutan inmediatamente al mejor precio disponible en el mercado.
- **Órdenes Stop-Loss:** Se utilizan para limitar las pérdidas potenciales en una posición. Si el precio alcanza el nivel de stop-loss, se activa una orden de mercado para cerrar la posición.
- **Órdenes Take-Profit:** Se utilizan para asegurar ganancias cuando el precio alcanza un objetivo predefinido. Si el precio alcanza el nivel de take-profit, se activa una orden de mercado para cerrar la posición.

5.6.2. Lógica de Gestión de Riesgos

El bot permite al usuario configurar porcentajes de stop-loss y take-profit. Estos se calculan en función del precio de entrada de la operación y se ajustan dinámicamente.

Capítulo 6

Implementación

Este capítulo detalla la implementación técnica del bot de trading, cubriendo el entorno de desarrollo, la estructura del proyecto y los aspectos clave de cada módulo.

6.1. Entorno de Desarrollo

- **Lenguaje de Programación:** Python 3.9+
- **Bibliotecas Principales:**
  - Tkinter: Para la interfaz gráfica.
  - CCXT: Para la interacción con APIs de exchanges.
  - Pandas: Para la manipulación de datos y cálculo de indicadores.
  - Matplotlib y mplfinance: Para la generación de gráficos.
  - Threading: Para la gestión de la concurrencia.
- **Entorno Virtual:** Se recomienda el uso de un entorno virtual (e.g., venv) para gestionar las dependencias del proyecto.
- **Control de Versiones:** Git y GitHub para el seguimiento de cambios y colaboración.

6.2. Estructura del Proyecto

El proyecto se organiza en los siguientes archivos y directorios principales:

- `main.py`: Punto de entrada de la aplicación, contiene la clase principal `CryptoApp` que gestiona la GUI y la lógica general.
- `ui_styles.py`: Define los componentes personalizados de la interfaz gráfica moderna.

- `conexion.py`: Módulo para la gestión de conexiones con los exchanges (`ExchangeManager`).
- `chart.py`: Módulo para la gestión de gráficos (`ChartManager`).
- `cartera.py`: Módulo para la gestión de la cartera (`CarteraManager`).
- `transacciones.py`: Módulo para la gestión de transacciones.
- `pnl_tracker.py`: Módulo para el seguimiento del PnL (`PnLTracker`).
- `calculos.py`: Módulo con funciones para cálculos técnicos (SMAs, etc.).
- `config.py`: Archivo para almacenar claves API y configuraciones (no incluido en el control de versiones).
- `logs/`: Directorio para almacenar archivos de log.
- `images/`: Directorio para almacenar imágenes utilizadas en la GUI o en la documentación.

6.3. Conexión con Exchanges (conexion.py)

El módulo `conexion.py` implementa la clase `ExchangeManager`, responsable de establecer y gestionar las conexiones con Binance y Bitfinex utilizando la biblioteca CCXT. Proporciona métodos unificados para:

- Autenticar con claves API.
- Obtener datos de mercado (velas OHLCV, libro de órdenes, ticker).
- Consultar saldos de cuenta.
- Crear, cancelar y consultar el estado de las órdenes.

6.4. Implementación de la Interfaz Gráfica (main.py, ui_styles.py)

La clase principal `CryptoApp` en `main.py` inicializa la ventana raíz de Tkinter y construye la interfaz gráfica utilizando los componentes definidos en `ui_styles.py`.

6.4.1. Panel de Control Izquierdo

Contiene los selectores de exchange, par, timeframe, botones de conexión, y controles para iniciar/detener la estrategia.

6.4.2. Panel de Datos Derecho

Utiliza un `ttk.Notebook` para organizar las pestañas de Gráfico, Transacciones, PnL y Logs.

6.5. Implementación de la Estrategia de Trading (calculos.py, main.py)

La lógica para calcular las SMAs se encuentra en `calculos.py`. La clase `CryptoApp` en `main.py` implementa el bucle principal de la estrategia que:

1. Obtiene los datos de velas más recientes.
2. Calcula las SMAs corta y larga.
3. Comprueba si hay un cruce de SMAs.
4. Si se detecta una señal, genera una orden de compra o venta.
5. Aplica la lógica de stop-loss y take-profit a las posiciones abiertas.

6.6. Gestión de Órdenes (transacciones.py)

El módulo `transacciones.py` se encarga de la lógica para crear y enviar los diferentes tipos de órdenes (mercado, stop-loss, take-profit) a través del `ExchangeManager`. Este componente centraliza toda la gestión de transacciones del sistema y actúa como intermediario entre la interfaz de usuario y los exchanges.

Las funcionalidades principales incluyen:

- **Creación de órdenes manuales**: Proporciona una interfaz gráfica intuitiva donde el usuario puede seleccionar monedas, especificar cantidades y confirmar transacciones con validación en tiempo real de saldos y precios
- **Órdenes automáticas**: Ejecuta transacciones generadas por las estrategias algorítmicas, manejando los parámetros predefinidos sin intervención del usuario
- **Gestión de stop-loss y take-profit**: Implementa la lógica específica para cada exchange, utilizando `STOP_LOSS_LIMIT` en Binance y órdenes de tipo `stop` en Bitfinex
- **Validación y precisión**: Controla los montos mínimos de operación y ajusta la precisión de precios según las especificaciones técnicas de cada exchange
- **Registro automático**: Se integra con el sistema PnL para registrar automáticamente todas las operaciones y mantener un historial completo

- **Manejo robusto de errores**: Gestiona situaciones como pérdida de conectividad, saldo insuficiente, parámetros inválidos y limitaciones específicas de cada exchange

El módulo también maneja las diferencias entre exchanges, como el uso de pares USD/USDT en Bitfinex versus USDT en Binance, y aplica las reglas de precisión específicas de cada plataforma.

6.7. Seguimiento de Rendimiento (pnl_tracker.py)

El `PnLTracker` calcula el beneficio o pérdida de cada operación cerrada y mantiene un registro del PnL acumulado. Este módulo proporciona un análisis completo del rendimiento de trading con capacidades avanzadas de seguimiento y visualización.

Sus características principales son:

- **Registro persistente**: Almacena todas las transacciones en archivos JSON organizados por exchange y categoría (general o estrategia específica), garantizando que los datos se conserven entre sesiones
- **Cálculo de PnL realizado**: Rastrea las ganancias y pérdidas de operaciones completadas utilizando el método de promedio de costos para calcular el beneficio real de cada venta
- **PnL no realizado**: Evalúa el valor actual de las posiciones abiertas consultando los precios actuales del mercado para mostrar ganancias/pérdidas potenciales
- **Métricas avanzadas**: Calcula automáticamente el ROI (Return on Investment), número total de operaciones, ratio de operaciones ganadoras/perdedoras y otros indicadores de rendimiento
- **Visualización gráfica**: Genera gráficos de PnL acumulado utilizando matplotlib, integrados directamente en la interfaz gráfica para análisis visual de tendencias
- **Historial detallado**: Presenta una tabla interactiva con todas las transacciones, incluyendo fechas, precios, cantidades y beneficios específicos de cada operación

El sistema separa los datos por exchange y permite análisis independiente del rendimiento en Binance y Bitfinex, facilitando la comparación de resultados entre plataformas.

6.8. Sistema de Logging

Se implementa un sistema de logging personalizado que registra eventos importantes, decisiones de la estrategia, errores y operaciones tanto en la interfaz gráfica como en la consola. El sistema utiliza un enfoque híbrido que combina logging visual en tiempo real con registro de consola para debugging.

Las características del sistema incluyen:

- **Logging visual integrado**: Utiliza un widget `Text` de Tkinter incorporado en la pestaña de estrategias que muestra eventos en tiempo real con auto-scroll automático y timestamps precisos
- **Seguridad de threading**: Implementa actualizaciones thread-safe utilizando `root.after()` para evitar conflictos cuando los hilos de estrategia intentan actualizar la interfaz gráfica
- **Categorización de eventos**: Diferencia entre tipos de eventos como inicio/fin de estrategias, ejecución de órdenes, detección de señales de trading, errores de sistema y cálculos de P&L
- **Formato estructurado**: Aplica timestamps automáticos, utiliza iconos distintivos (, ,) y mantiene un formato consistente para facilitar la lectura y análisis de logs
- **Logging de estrategias**: Registra detalladamente los parámetros de configuración, señales detectadas por los indicadores técnicos, ejecución de órdenes stop-loss y take-profit
- **Gestión específica de errores**: Proporciona logging detallado para errores de conexión con exchanges, problemas de validación de parámetros y fallos en la ejecución de órdenes

El sistema facilita significativamente el debugging del bot, permite monitoreo en tiempo real del comportamiento de las estrategias y mantiene un histórico completo de actividades para análisis posterior del rendimiento y detección de patrones.

Capítulo 7

Resultados

Este capítulo presenta los resultados obtenidos durante las pruebas del bot de trading, incluyendo la configuración del entorno de pruebas, la validación funcional y ejemplos de operativa.

7.1. Entorno de Pruebas

Las pruebas se realizaron principalmente en el entorno de paper trading de Bitfinex para simular operaciones sin riesgo financiero. También se realizaron pruebas de conexión y obtención de datos con Binance.

- **Plataformas:** Bitfinex (Paper Trading), Binance (Datos).
- **Pares de Trading:** ETH/BTC, USDT/USD.
- **Timeframes:** 1m, 5m, 15m, 1h.
- **Períodos SMA Típicos:** Corta (e.g., 10, 20, 25), Larga (e.g., 30, 50, 100).

7.2. Pruebas Funcionales

Se realizaron pruebas exhaustivas para validar cada funcionalidad del bot:

- Conexión y desconexión con los exchanges.
- Obtención y visualización correcta de datos de mercado y gráficos.
- Cálculo preciso de indicadores SMA.
- Generación de señales de compra/venta según la estrategia de cruce de SMAs.
- Ejecución de órdenes de mercado, stop-loss y take-profit.
- Actualización de la cartera y cálculo del PnL.

- Funcionamiento del sistema de logging.
- Responsividad y usabilidad de la interfaz gráfica.

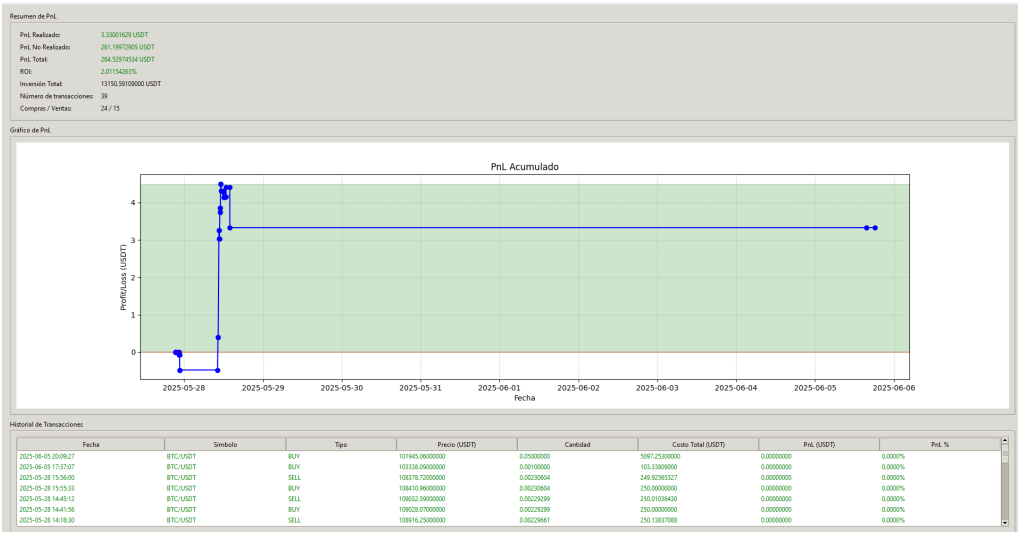

Figura 7.1: Análisis PnL de Binance, con las compras y ventas y la evolución con el paso del tiempo

7.3. Ejemplos de Operativa y Casos de Uso

7.3.1. Caso de Uso 1: Detección de Cruce Alcista y Compra

El bot monitoriza continuamente los datos de mercado y detecta cuando la SMA corta cruza por encima de la SMA larga, generando una señal de compra. En ese momento, ejecuta automáticamente una orden de compra a precio de mercado y establece órdenes stop-loss y take-profit para gestionar el riesgo.

7.3.2. Caso de Uso 2: Detección de Cruce Bajista y Venta

Cuando la SMA corta cruza por debajo de la SMA larga, el bot genera una señal de venta. Si hay una posición abierta, el bot ejecuta una orden de venta a precio de mercado para cerrar la posición y asegurar las ganancias o limitar las pérdidas.

7.3.3. Caso de Uso 3: Configuración y Activación de Stop-Loss

El bot permite configurar órdenes stop-loss para limitar las pérdidas potenciales. Estas órdenes se activan automáticamente cuando el precio alcanza el nivel predefinido.

Figura 7.2: Gráfico BTC/USDT con velas de 5 minutos y SMA corto de 25 (rojo) y largo de 50 (morado) mostrando un cruce alcista

7.3.4. Caso de Uso 4: Ejecución de Órdenes de Mercado y Take-Profit

El bot puede ejecutar órdenes de mercado inmediatas y configurar órdenes take-profit para asegurar ganancias cuando el precio alcanza un objetivo predefinido.

7.4. Limitaciones

- La estrategia de cruce de SMAs es simple y puede no ser rentable en todas las condiciones de mercado (e.g., mercados laterales).
- El rendimiento del bot depende de la correcta configuración de los parámetros (períodos de SMA, niveles de stop-loss/take-profit).
- No se implementan mecanismos avanzados de gestión de riesgos más allá de stop-loss y take-profit por operación.
- El bot no considera factores fundamentales ni noticias del mercado.
- La robustez ante fallos de red o del exchange podría mejorarse con lógicas de reintento más sofisticadas.

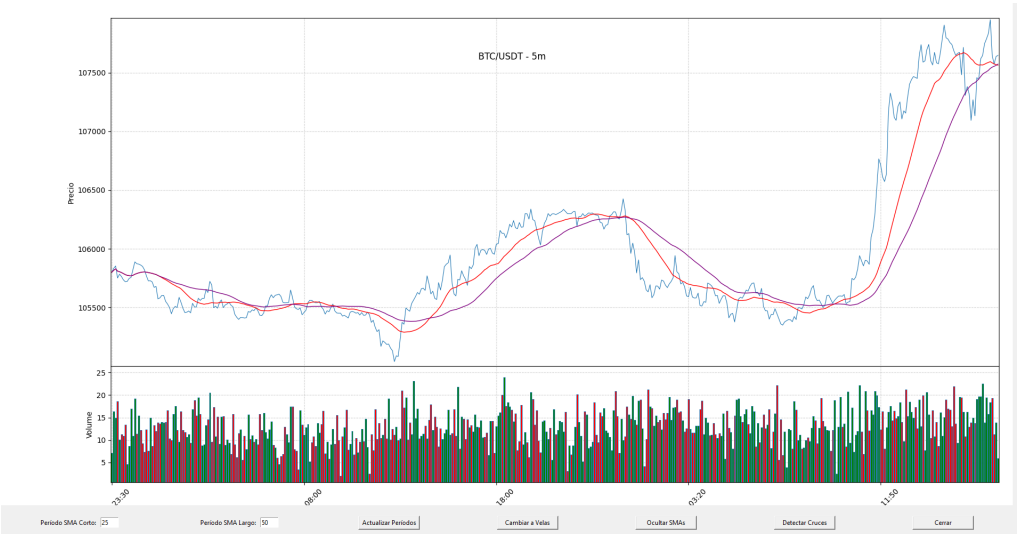

Figura 7.3: Gráfico BTC/USDT con línea de 5 minutos y SMA corto de 25 (rojo) y largo 50 (morado) mostrando un cruce bajista

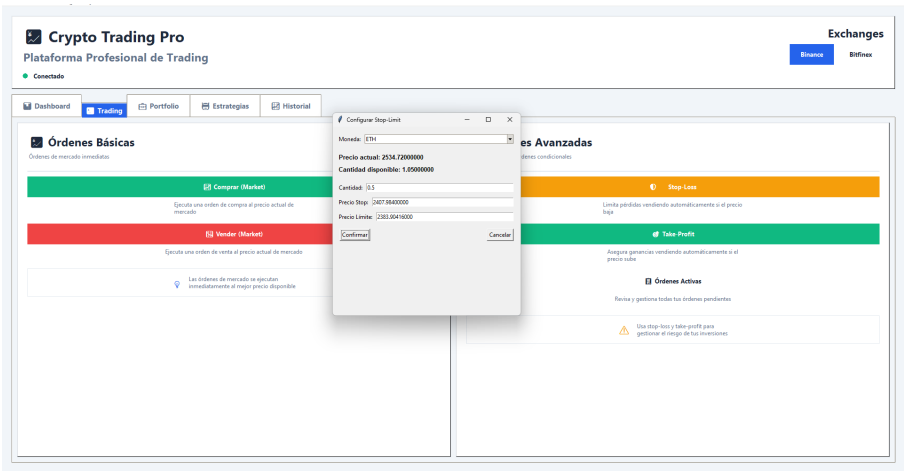

Figura 7.4: Establecer orden Stop-Limit en ETH

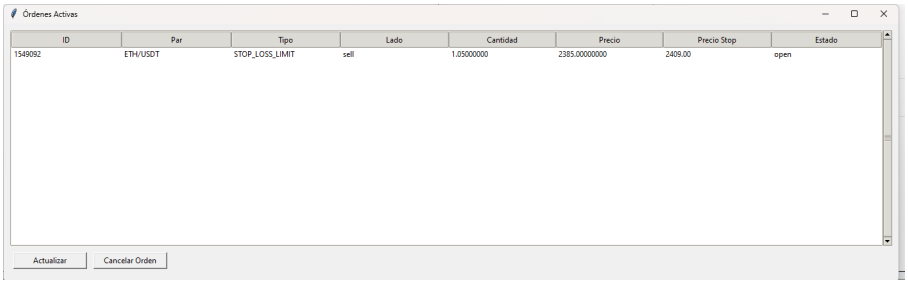

Figura 7.5: Visualización de la orden stop-limit puesta para ETH

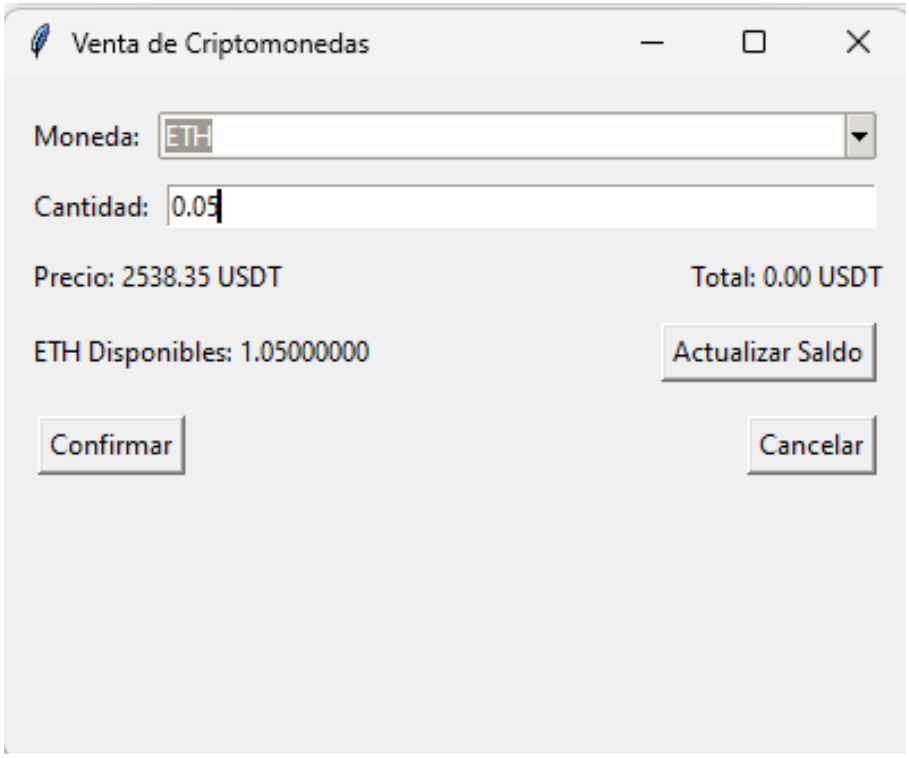

Figura 7.6: Venta de 0.05 ETH a precio de mercado

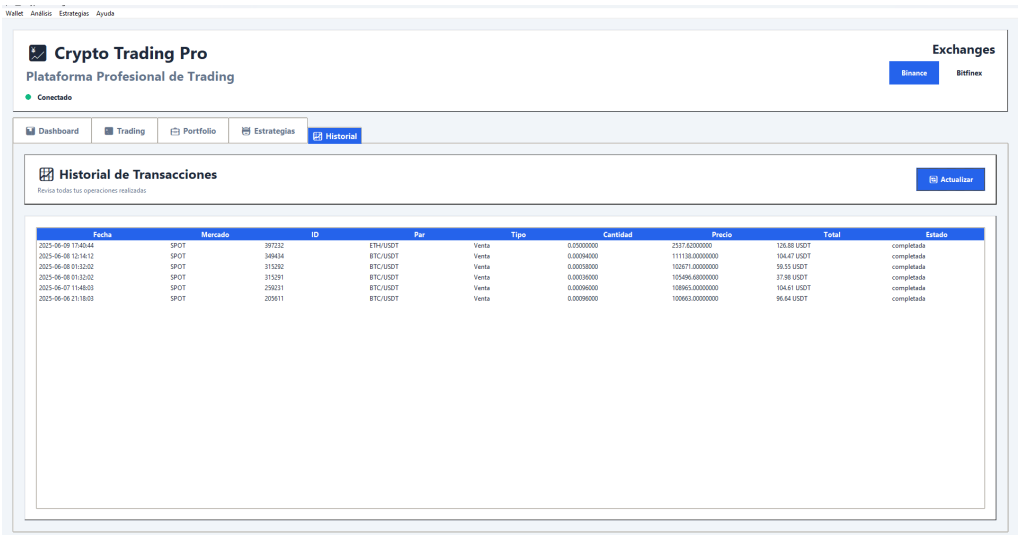

Figura 7.7: Historial de órdenes y transacciones de Binance, con la orden de venta antes mencionada de ETH-USDT

Capítulo 8

Conclusiones

8.1. Resumen del Proyecto y Cumplimiento de Objetivos

Este Trabajo Final de Grado ha abordado con éxito el diseño, desarrollo e implementación de un bot de trading de criptomonedas basado en análisis técnico. Se han cumplido los objetivos principales, incluyendo la creación de un bot funcional capaz de conectarse a los exchanges Binance y Bitfinex, implementar la estrategia de cruce de Medias Móviles Simples (SMA), gestionar órdenes con stop-loss y take-profit, y proporcionar una interfaz gráfica de usuario intuitiva para su operación y monitorización.

El bot permite la automatización de decisiones de trading para los pares ETH/BTC y USDT/USD, demostrando la viabilidad de aplicar conocimientos de ingeniería informática en el dinámico campo de las finanzas digitales. La utilización de herramientas como Python, Tkinter y la biblioteca CCXT ha facilitado un desarrollo modular y eficiente.

8.2. Reflexiones sobre la Estrategia y el Desarrollo

La estrategia de cruce de SMAs, aunque fundamental, ha servido como un excelente punto de partida para explorar los mecanismos del trading algorítmico. Su implementación ha permitido comprender los desafíos asociados con la detección de señales, la ejecución de órdenes y la gestión de riesgos en un entorno de alta volatilidad.

El desarrollo de la interfaz gráfica moderna ha sido un aspecto importante, buscando un equilibrio entre funcionalidad y usabilidad. La modularidad de la arquitectura ha demostrado ser beneficiosa, permitiendo aislar componentes y facilitar tanto el desarrollo como las pruebas.

Los principales retos técnicos, como la gestión de la concurrencia, el manejo de errores de API y la sincronización de datos, se han abordado con soluciones prácticas, aunque siempre existe margen para la optimización y el refinamiento.

8.3. Trabajo Futuro

El bot desarrollado sienta las bases para numerosas mejoras y extensiones futuras:

- **Incorporación de Más Estrategias:** Añadir otras estrategias de análisis técnico (e.g., RSI, MACD, Bandas de Bollinger) o incluso basadas en aprendizaje automático.
- **Soporte para Más Exchanges y Pares:** Extender la compatibilidad a otros exchanges populares y una gama más amplia de criptomonedas.
- **Optimización de Estrategias y Backtesting Avanzado:** Implementar un módulo de backtesting robusto que permita optimizar los parámetros de las estrategias utilizando datos históricos.
- **Mejoras en la Gestión de Riesgos:** Incorporar técnicas más sofisticadas de gestión de riesgos, como el dimensionamiento de posiciones basado en la volatilidad o el capital disponible.
- **Notificaciones y Alertas:** Añadir un sistema de notificaciones (e.g., por correo electrónico o Telegram) para alertar al usuario sobre eventos importantes.
- **Despliegue en Servidor:** Adaptar el bot para su ejecución continua en un servidor en la nube.
- **Análisis de Sentimiento de Mercado:** Integrar fuentes de datos externas para análisis de sentimiento de noticias o redes sociales.
- **Mejoras en la Interfaz Gráfica:** Continuar refinando la GUI con más visualizaciones interactivas y opciones de personalización.

Este proyecto ha proporcionado una valiosa experiencia en el desarrollo de software aplicado a los mercados financieros y abre un abanico de posibilidades para futuras investigaciones y desarrollos en el apasionante mundo del trading algorítmico de criptomonedas.

Índice de figuras

Índice de cuadros

Bibliografía

[1] Binance. *Binance API Documentation*. 2025. URL: https://developers.binance.com/docs/binance-spot-api-docs/rest-api (visitado 06-06-2025).

[2] Bitfinex. *Bitfinex API Documentation*. 2025. URL: https://docs.bitfinex.com/docs/introduction (visitado 06-06-2025).

[3] Will Kenton. *Simple Moving Average (SMA): What It Is and the Formula*. 2024. URL: https://www.investopedia.com/terms/s/sma.asp (visitado 06-06-2025).

[4] Perry J. Kaufman. *Trading Systems and Methods*. 5.ª ed. Wiley, 2013. ISBN: 978-1118043561.

[5] Python Software Foundation. *Tkinter - Python interface to Tcl/Tk*. 2025. URL: https://docs.python.org/3/library/tkinter.html (visitado 06-06-2025).

[6] Ernest P. Chan. *Algorithmic Trading: Winning Strategies and Their Rationale*. Wiley, 2013. ISBN: 978-1118460146.

[7] CCXT. *CCXT - CryptoCurrency eXchange Trading Library Documentation*. 2025. URL: https://docs.ccxt.com/ (visitado 06-06-2025).

Anexo A: Guía de Instalación y Configuración

Esta sección proporciona una guía detallada para la instalación y configuración del bot de trading de criptomonedas desarrollado. El sistema está implementado en `Python` y requiere la instalación de varias bibliotecas específicas, así como la configuración adecuada de las APIs de los exchanges.

A.1. Requisitos del Sistema

El sistema ha sido desarrollado y probado en los siguientes entornos:

- **Python:** Versión 3.8 o superior
- **Sistema Operativo:** Windows 10/11, macOS, Linux
- **RAM:** Mínimo 4GB recomendado
- **Conexión a Internet:** Estable para comunicación con exchanges

A.2. Instalación de Dependencias

Bibliotecas Principales

```
# Biblioteca para conectividad con exchanges
pip install ccxt

# Bibliotecas para interfaz gráfica
pip install tkinter  # Generalmente incluida con Python
pip install PyQt6

# Procesamiento de imágenes
pip install Pillow

# Bibliotecas para gráficos y análisis
pip install matplotlib
pip install mplfinance
pip install pandas

# Bibliotecas de red
pip install requests
```

Listado 1: Instalación de bibliotecas principales

Comando de Instalación Unificado

```
pip install ccxt PyQt6 Pillow matplotlib mplfinance pandas
   requests
```

Listado 2: Instalación completa de dependencias

Verificación de la Instalación

```
import sys

try:
    import ccxt
    print("CCXT instalado correctamente")
except ImportError:
    print("CCXT no está instalado")

try:
    from PyQt6.QtWidgets import QApplication
    print("PyQt6 instalado correctamente")
except ImportError:
    print("PyQt6 no está instalado")

```

```
try:
    from PIL import Image
    print("Pillow instalado correctamente")
except ImportError:
    print("Pillow no está instalado")

try:
    import matplotlib.pyplot as plt
    import mplfinance as mpf
    print("Matplotlib y mplfinance instalados correctamente")
except ImportError:
    print("Matplotlib o mplfinance no están instalados")

try:
    import pandas as pd
    print("Pandas instalado correctamente")
except ImportError:
    print("Pandas no está instalado")

try:
    import requests
    print("Requests instalado correctamente")
except ImportError:
    print("Requests no está instalado")

print(f"Versión de Python: {sys.version}")
```

Listado 3: Script de verificación de dependencias

A.3. Configuración de las APIs

El bot requiere la configuración de claves API para los exchanges Binance y Bitfinex. Esta configuración se realiza en el archivo `config.py`.

Obtención de Claves API

Binance (Modo Sandbox)

1. Registrarse en Binance Testnet: https://testnet.binance.vision/
2. Acceder a la sección `API Management`
3. Crear una nueva API Key con permisos:
   - `Enable Reading`
   - `Enable Spot & Margin Trading`
   - `Enable Futures`
4. Copiar las claves generadas

Bitfinex (Modo Producción)

1. Registrarse en Bitfinex: https://www.bitfinex.com/
2. Ir a `Account >API`
3. Crear nueva API Key con:
   - `Orders: create and cancel`
   - `Positions: view`
   - `Wallets: view balances`
   - `Margin info: view`
4. Activar restricciones de IP si es necesario

Edición del Archivo `config.py`

```
# Configuración de APIs
API_BINANCE = "tu_api_key_de_binance_testnet"
CLAVE_SECRETA_BINANCE = "tu_secret_key_de_binance_testnet"
API_BITFINEX = "tu_api_key_de_bitfinex"
CLAVE_SECRETA_BITFINEX = "tu_secret_key_de_bitfinex"

# Configuración de la interfaz
COLORES = {
    'conectado': 'green',
    'desconectado': 'red',
    'error': 'red',
    'exito': 'green'
}

# Actualización automática
INTERVALO_ACTUALIZACION = 20  # en segundos

# Exchanges disponibles
EXCHANGES = {
    'binance': {
        'nombre': 'Binance',
        'logo_path': 'Binance_Logo.svg.png',
        'logo_size': (40, 40),
        'color': '#000000'
    },
    'bitfinex': {
        'nombre': 'Bitfinex',
        'logo_path': 'Bitfinex-Logo.png',
        'logo_size': (40, 40),
        'color': '#FFFFFF'
    }
```

```
}
```

Listado 4: Configuración de APIs en config.py

A.4. Estructura del Proyecto

```
crypto-trading-bot/
          main.py                    # Punto de entrada de la
   app
          config.py                  # Parámetros y claves API
          conexion.py                # Gestión de exchanges
          cartera.py                 # Gestión de activos
          transacciones.py           # Registro de operaciones
          chart.py                   # Análisis técnico
          ui_styles.py               # Estilos de la interfaz
          calculos.py                # Lógica de cálculo
          pnl_tracker.py             # Seguimiento de P&L
          Binance_Logo.svg.png       # Logo de Binance
          Bitfinex-Logo.png          # Logo de Bitfinex
          data/                      # Carpeta de almacenamiento
```

Listado 5: Estructura de archivos del proyecto

A.5. Ejecución del Bot

```
python main.py
```

Listado 6: Ejecución del bot

Verificación Inicial

- La interfaz debe abrirse sin errores
- El estado debe iniciar como "Desconectado"
- Los botones de conexión deben estar activos
- No debe haber errores en consola

A.6. Resolución de Problemas Comunes

Error: PyQt6 no instalado

```
pip install PyQt6
# O usando Conda:
conda install pyqt
```

Listado 7: Solución PyQt6

Error de conexión con la API

- Verifica las claves API
- Revisa los permisos habilitados
- Asegura uso de testnet en Binance
- Comprueba la conexión a Internet

Problemas con matplotlib

```
pip install matplotlib --upgrade
pip install mplfinance --upgrade
```

Listado 8: Reinstalación de matplotlib

A.7. Configuración Avanzada

Parámetros Personalizables en `config.py`

- `INTERVALO_ACTUALIZACION`: Intervalo entre actualizaciones
- `COLORES`: Estado visual de la interfaz
- Detalles visuales por exchange

Modos de Operación

- **Binance:** Usa modo sandbox por defecto
- **Bitfinex:** Producción (se recomienda cuidado)
- Modifica el parámetro `test=True` en `conexion.py` para cambiar el modo

Esta guía cubre todos los aspectos esenciales para desplegar y utilizar correctamente el bot desarrollado para este trabajo de fin de grado.