UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LETÍCIA D SANTI MOREIRA

PROPOSTA DE UM SISTEMA DE ENTREGA COM VEÍCULOS AÉREOS NÃO TRIPULADOS BASEADO EM BLOCKCHAIN E SMART CONTRACT

PATO BRANCO

2022

LETÍCIA D SANTI MOREIRA

PROPOSTA DE UM SISTEMA DE ENTREGA COM VEÍCULOS AÉREOS NÃO TRIPULADOS BASEADO EM BLOCKCHAIN E SMART CONTRACT

A delivery system proposal with unmanned aerial vehicles based on Blockchain and Smart Contract

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharela em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Jeferson José de Lima

PATO BRANCO

2022

LETÍCIA D SANTI MOREIRA

PROPOSTA DE UM SISTEMA DE ENTREGA COM VEÍCULOS AÉREOS NÃO TRIPULADOS BASEADO EM BLOCKCHAIN E SMART CONTRACT

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharela em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de Aprovação: 20 de junho de 2022.

Prof. Dr. Jeferson José de Lima
Universidade Tecnológica Federal do Paraná

Prof. Dr. Adriano Serckumecka
Universidade Tecnológica Federal do Paraná

Profa. Dra. Luciene De Oliveira Marin
Universidade Tecnológica Federal do Paraná

Prof. Dr. Marcelo Teixeira
Universidade Tecnológica Federal do Paraná

PATO BRANCO

2022

RESUMO

MOREIRA, Letícia. Proposta de um sistema de entrega com veículos aéreos não tripulados baseado em Blockchain e *Smart Contract*. 2022. 46 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2022.

Este trabalho apresenta uma proposta de arquitetura e desenvolvimento de um sistema de entrega com veículos aéreos não tripulados baseado em Blockchain e *Smart Contract*. Neste sistema é integrado a Blockchain Ethereum, o ambiente de simulação Gazebo e a controladora de voo PX4. A implementação do sistema está divida em três aplicações desenvolvidas na linguagem Pyhton, uma para a escrita das funções em Solidity do *Smart Contract* e sua implantação na Blockchain, outra para a integração do *Smart Contract* e o veículo aéreo não tripulado e finalmente uma para a solicitação de entrega através de escritas no *Smart Contract*. A PX4 é simulada com a técnica *software-in-the-loop*, recebendo missões de voo referente as entregas e controlando o veículo até o destino. Para a emulação local da Blockchain Ethereum utiliza-se a ferramenta Ganache CLI e para a simulação de entrega no Gazebo é utilizado o mapa da região do aeroporto de *San Carlos* nos Estados Unidos, com o veículo aéreo não tripulado de modelo Thypon H480.

Palavras-chave: Blockchain; *Smart Contract*; Veículo aéreo não tripulado; Integração; Aplicação.

ABSTRACT

MOREIRA, Letícia. Proposta de um sistema de entrega com veículos aéreos não tripulados baseado em Blockchain e *Smart Contract*. 2022. 46 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2022.

This final paper presents an architecture proposal and development of delivery system with unmanned aerial vehicles based on Blockchain and Smart Contract.This system integrates the Ethereum Blockchain, the simulation environment Gazebo and the flight controller PX4.The system implementation is divided into three applications developed using Python programming language. One aplication is used to define the Smart Contract in Solidity and deploy it in the Blockchain. The second one integrates Smart Contract and the unmanned aerial vehicle. The last one allows the delivery requests to the Smart Contract. The PX4 is simulated using the software-in-the-loop technique, receiving flight missions for deliveries and controlling the vehicle to delivery point. The Ganache CLI tool is used to emulate a local Ethereum Blockchain. Gazebo is used to simulate the deliveries inside the San Carlos airport environment, United States, and the unmanned aerial vehicle, Thypon H480.

Keywords: Blockchain; Smart Contract; Unmanned Aerial Vehicle; Integration; Application.

LISTA DE CÓDIGOS-FONTE

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

SIGLAS

API	*Application Programming Interface*
B2B	*Business-to-business*
B2C	*Business-to-consumer*
DDoS	*Distributed Denial of Service*
EVM	*Ethereum Virtual Machine*
P2P	*Peer-to-peer*
PoS	*Proof-of-Stake*
PoW	*Proof-of-Work*
SaaS	*Software as a Service*
VANT	Veiculo Aéreo não tripulado
VANTs	Veiculos Aéreos não tripulados

SUMÁRIO

1 INTRODUÇÃO

Devido a sua versatilidade, flexibilidade, fácil instalação e custo operacional relativamente baixo, os veículos aéreos não tripulados (VANTs) são vistos como promissores tanto para uso civil como para uso militar (BEKMEZCI; SAHINGOZ; TEMEL, 2013). Durante os últimos anos, seu uso em aplicações como monitoramento de tráfego em rodovias, sensoreamento remoto, auxílio em operações de resgates, agricultura de precisão, segurança, vigilância e entregas está crescendo (SHAKHATREH *et al.*, 2019). O mercado global de VANTs em 2020 foi avaliado em $4.46 bilhões e as projeções indicam quer irá valer $8.55 bilhões em 2030 (HIMANSHU JOSHI, 2021).

Na área de logística, a utilização de VANTs pode diminuir o tempo da entrega, custo e torna-lá mais ecológica se comparadas às entregas tradicionais (YOO; YU; JUNG, 2018). A empresa irlandesa de entregas por VANTs, Manna, mostrou que ao utilizar os veículos aéreos tem-se uma economia nos custos de 90% se comparada aos métodos de entregas baseados em carro como o Uber Eats e também tem impacto ambiental 50 vezes mais eficiente (KOETSIER, 2021).

Conforme Otto *et al.* (2018) várias indústrias podem se beneficiar das tecnologias de aeronaves que não necessitam de um piloto diminuindo o custo com trabalhadores e possibilitando operações em ambientes perigosos aos seres humanos. Essas aeronaves não necessitam de infraestrutura rodoviária e podem alcançar locais inacessíveis a veículos terrestres, além disso, por não possuírem piloto a bordo o peso da aeronave é menor acabando por consumir menos energia. Também é possível diminuir a quantidade de erros humanos e otimizar o processo de entrega com um sistema automatizado, porém ao ter todos os membros do sistema de logística interconectados ele estará sujeito a *cyberattacks*, vírus, a *bug* de *software* e *hardware* a seus componentes (CHEUNG; BELL, 2019).

Vários pesquisadores abordam tais tecnologias, Ling e Draghic (2019) apresentam um caso de estudo para um sistema de entrega de plasma, plaquetas e bolsas de sangue por VANTs com objetivo de atender locais remotos e diminuir o tempo de entrega, Claesson *et al.* (2017) propõe um sistema para envio de um desfibrilador automático externo, Yong Sik Chang e Hyun Jung Lee (2018) sugere que caminhões e VANTs trabalhem em conjunto para otimizar o tempo da entrega.

O modelo tradicional utilizado para desenvolver uma aplicação é centralizado, utilizando um único ponto para armazenar e transmitir informações. Segundo Strawn (2019), o modelo

centralizado cliente-servidor está sujeito a diversas falhas de segurança, como o *Distributed Denial of Service* (DDoS), sendo este um ataque de negação de serviço distribuído em que o servidor recebe um número elevado de requisições e pode ficar indisponível. Esse modelo também pode ter problemas com sincronização e baixo desempenho afetando as partes envolvidas em uma transação (CHANG, S. E.; CHEN; LU, 2019). Uma tecnologia que permite armazenar dados de forma descentralizada é a Blockchain. Geralmente é adotado o protocolo P2P para garantir que os computadores em uma rede são iguais entre si sem um nó especial. Por não possuir um servidor centralizado uma rede P2P é resistente a ataques e altamente tolerável a falhas (WANG; SU, 2020).

1.1 OBJETIVOS

O objetivo geral deste trabalho é desenvolver um sistema descentralizado com Blockchain e *Smart Contract* que gerencia e armazena as solicitações de entregas para um veículo aéreo não tripulado.

1.1.1 Objetivos Específicos

- Definir e avaliar as regras de negócios entre Usuário e Administrador do VANT.
- Implementar o *Smart Contract* na linguagem de programação Solidity e implantá-lo em uma Blockchain.
- Desenvolver uma aplicação que integre a unidade de voo autônomo do VANT e a Blockchain.
- Elaborar uma aplicação que permita a solicitação de entregas pelo usuário.
- Configurar o cenário de operação em ambiente de simulação.
- Definir casos de uso para validar as funcionalidades a partir de simulações.

1.2 JUSTIFICATIVA

O mercado de entregas está crescendo e, devido ao aumento nas compras *on-line*, principalmente a demanda por entregas diretas ao consumidor. Foi percebido que o tempo de entrega é um fator importante para os consumidores sendo que uma grande parte estaria disposta a pagar mais por entregas realizadas no mesmo dia (JOERSS; NEUHAUS; SCHRÖDER, 2016).

Em agosto de 2020, a Agência Nacional de Aviação Civil (Anac) passou a permitir teste para entregas de produtos com veículos autônomos não tripulados no Brasil. A empresa Speedbird executa testes em campo com entregas para o iFood em Campinas (SP), o trajeto de teste é entre o Shopping Iguatemi e o iFood Hub que se encontra a 400 metros de distância. Esse percurso é realizado pelo VANT em dois minutos, trajeto que percorrido a pé leva 12 minutos (AGUIAR, 2021; FOGAÇA, 2021b,a).

O Wing, é o serviço de entregas por VANTs da Alphabet disponível na Austrália, Estados Unidos e Finlândia. As aeronaves do serviço entregam pacotes com menos de 2,3 Kg e em agosto de 2021 já contava com mais de 100 mil entregas efetuadas. Na Austrália o Wing passou a permitir que empresas loquem as aeronaves e façam entregas direto de suas instalações, sem precisar utilizar os posto de entrega da Wing (KOETSIER, 2021).

Conforme Keeney (2015), o preço de entrega estimado para o consumidor final de um pacote utilizando o serviço de entregas por VANTs em desenvolvimento pela Amazon, o Amazon Prime Air é de $0,88 sendo que o preço para o mesmo pacote utilizando o Amazon's Prime é $5,99 e $7,99 para o Amazon Prime Now. Já a empresa Flytrex possui um serviço de entregas por VANTs autônomos com entregas de pacotes de até 3 Kg para até 5 milhas e está disponível nos Estados Unidos. O usuário pode solicitar a entrega de produtos disponíveis em lojas cadastradas no aplicativo disponibilizado pela Flytrex (FLYTREX, 2021).

Com o aumento da demanda de entregas por compras realizadas pela *internet* é importante para o cliente e o vendedor que a entrega seja realizada de forma segura e transparente entre as partes, principalmente quando estão em localidades distantes e não possuem confiança entre si. Em uma entrega podem estar envolvidos diferentes empresas e categorias de transportes sendo que a responsabilidade de fornecer os dados de rastreio dessa entrega geralmente é dos funcionários da transportadora, fazendo com que os dados possam ser não confiáveis (HASAN; SALAH, 2018).

Alguns sistemas decentralizados para vendas baseados em Blockchain estão disponíveis, como o Syscoin Platform que fornece um sistema para a troca de produtos e até pequenos documentos digitais. Nesse sistema o vendedor e comprador concordam em envolver uma terceira parte responsável por validar o processo que será recompensada pela sua contribuição. É permitido ao vendedor incluir um vídeo do envio do pacote, esse vídeo tera sua *hash* incluída como detalhe da transação, podendo ser utilizado como prova em disputas futuras. Porém, os dados de todo o transporte não é incluso na cadeia (HASAN; SALAH, 2018).

Acrescentando a tecnologia Blockchain a um sistema de entrega, pretende-se agregar

mais segurança ao sistema e evitar erros humanos, pois a Blockchain usa *logs* ordenados e eventos para fornecer rastreabilidade e auditabilidade (HASAN; SALAH, 2018).

1.3 ESTRUTURA DO TRABALHO

Este trabalho inicia com uma descrição das características de veículos aéreos não tripulados bem como alguns sistemas de entregas automatizados que utilizam esses veículos sendo abordado também simuladores computacionais realísticos na Seção 2.1. Em seguida, na Seção 2.2 é apresentado a tecnologia Blockchain e *Smart Contract*, detalhado as suas características e exposto como se relacionam.

Durante o Capítulo 3 é descrito o funcionamento do sistema e as ferramentas utilizadas no seu desenvolvimento. No capítulo 4 é detalhada a arquitetura proposta para o sistema de entrega, elaboração do *Smart Contract*, integração entre o veículo e a Blockchain e os testes efetuados para a validação do sistema.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos sobre o *hardware* escolhido, suas aplicações e simulação. Posteriormente as definições relacionadas a Blockchain e *Smart Contracts*.

2.1 VEÍCULOS AÉREOS NÃO TRIPULADOS

Um veículo aéreo não tripulado (VANT) é definido pela Agência Nacional de Aviação Civil (ANAC) como uma aeronave empregada para fins não recreativos e que não tenha piloto a bordo (ANAC, 2012). Para Arantes (2019), um VANT pode ser composto por *frame*, hélice, motor, servo motor, controlador do motor e bateria em quantidades a variar conforme o modelo sendo disponibilizados em diversas formas, valores e tamanhos.

Há muito tempo VANTs são utilizados para fins militares demonstrando potencial para uso em entregas (STOLAROFF *et al.*, 2018). Seu uso para entrega de pacotes oferece várias vantagens como não são limitados pela infraestrutura e congestionamento das rodovias, podendo entregar pacotes mais rapidamente que caminhões e carros. As aeronaves podem voar sobre montanhas e outros locais de difícil acesso pela terra, como áreas rurais que possuem pouca infraestrutura de acesso terrestre para entregas. Entregas com VANTs devem ter uma redução no impacto ambiental visto que diminuem a necessidade de caminhões de entregas (LEE, H. L. *et al.*, 2016).

Uma proposta de sistema de entrega automatizada de pequenos pacotes é feita por Neto (2016) e denominada PostDrone University. Nessa solução os pacotes são entregues entre helipontos espalhado no campus da universidade por um VANT UAV K-263, modelo que também é proposto pelo autor.

Com outra proposta, a empresa Drone Delivery Canada possui um sistema comercial de entregas com VANTs de depósito para depósito, o FLYTE. A solução comercial é disponibilizada atualmente pela empresa como um *Software as a Service* (SaaS) no Canadá e como serviço gerenciado licenciado internacionalmente. Nele é disponibilizado o rastreamento do transporte em tempo real e fornecida integração do FLYTE por APIs. Em novembro de 2021 a empresa inaugurou um centro de distribuição em Vaughan no Canadá visando expandir seu mercado para outras soluções como B2B e B2C. Ela possui 3 modelos de veículos proprietários dispostos no Quadro 1 (CANADA, 2021).

Segundo Ferrag e Maglaras (2019) o problema principal no desenvolvimento de um

Quadro 1 – Especificações dos modelos da Drone Delivery Canada

Modelo	Carga (Kg)	Distancia (Km)	Peso (Kg)
Sparrow	4,5	30	25
Robin XL	11.3	60	80
condor	180	200	476

Fonte: (CANADA, 2021).

sistema de entregas baseado em VANTs não está na parte física, mas sim em garantir a segurança e a privacidade. Com acesso aos dados dos perfis dos usuários um invasor do sistema pode desviar as rotas dos pacotes ou inserir dados falsos.

Na próxima seção será especificado o *hardware* utilizado neste trabalho.

2.1.1 Especificação do Hardware

Segundo Neto (2016), o principal componente de um VANT é seu controlador de voo e estabilidade. A partir dos dados obtidos de sensores como acelerômetro ele realiza o controle do veículo utilizando o sistema atuador. Esse sistema atuador é constituído pelos motores, sendo que a velocidade, altura e rotação do VANT podem ser controladas a partir da velocidade desses motores (COLLOTTA; PAU; CAPONETTO, 2014).

É possível automatizar um veículo fazendo o uso de uma controladora com função de piloto automático. A PX4 é uma controladora de voo profissional mantida por uma comunidade de desenvolvedores mundiais do meio industrial e acadêmico. Pode ser utilizada para controlar várias categorias de veículos, entre elas a de veículos aéreos. Com a PX4 é possível definir o destino para ela controlar veículo até o ponto, não sendo necessário um piloto externo para fazer o voo (PX4, 2021b).

Até o momento, a controladora de voo Pixhawk 4 mostrada na Figura 1 é a última versão de controladores da família Pixhawk executando o *software* do piloto automático PX4 em seu sistema embarcado Apache NuttX. Dispõe de um processador principal STM32F765 de 32 bits, arquitetura Arm Cortex-M7, memória de 2 megabytes e memória RAM de 512 Kilobytes. Ela também possui acelerômetro, giroscópio, magnetrômetro, barômetro e GPS (PX4, 2021b).

Outro fator importante em um veículo aéreo é a duração do voo e para maior duração é necessário escolher modelos com mais motores como hexacóptero e octocóptero. Com uma quantidade maior de motores sua potência, sustentação, carga útil, custo, peso e tamanho também aumentam. Outra vantagem é que esses modelos conseguem lidar com falhas de motores, em

Figura 1 – Controladora Pixhawk 4

Fonte: (PX4, 2021c).

caso de falha em um dos motores, os outros conseguirão compensar (ALAIMO *et al.*, 2013). Por esses motivos optou-se por utilizar neste trabalho um hexacóptero de modelo Typhoon H 480 da marca Yuneec. Quando não se dispõe do equipamento físico é possível simular o modelo do veículo para desenvolvimento e testes.

2.1.2 Simulador de Voo e Entrega

Segundo Arantes (2019), simuladores de voos tentam representar a dinâmica e as características de um sistema real através de um *software*. A NASA utiliza um simulador de voo nas missões de exploração em Marte, esse simulador utiliza o *software* de controle real de voo em suas simulações testando o comportamento real do sistema de robótica em operações na superfície e o comportamento da telecomunicação em simulações de operações (VERMA; LEGER, 2019). O sistema proposto nesse trabalho também será validado por simulações.

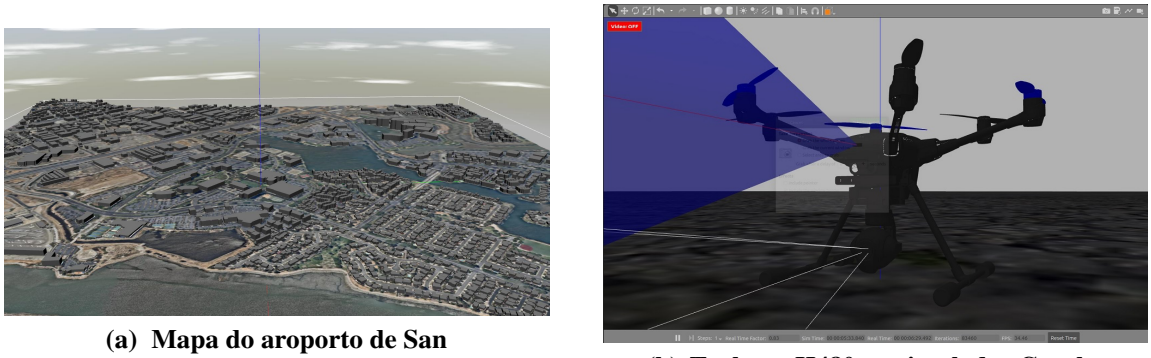
Na simulação, os objetos possuem massa, velocidade, fricção e outras características que fazem com que eles se comportem da mesma forma que em ambientes reais para quando são arrastados, puxados, esfregados, derrubados ou carregados. Assim um programa cliente vê uma interface para a simulação idêntica a do ambiente real sendo muito útil no processo de desenvolvimento de um sistema robótico (KOENIG; HOWARD, 2004).

Pode-se testar um sistema sob condições realísticas sem montar o *hardware* físico usando a técnica do *software-in-the-loop*, essa técnica é útil quando o custo para montá-lo é grande,

demorado ou não se tem o *hardware* final disponível (ISERMANN; SCHAFFNIT; SINSEL, 1999). Ela permite integrar o componente de *software* com um ambiente simulado (AYED; ZOUARI; ABID, 2017). Os autores Meola, Iannelli e Glielmo (2013) aplicam a técnica para validação do sistema de controle de um VANT de pequeno porte na presença de uma perturbação referente a um vento de 10 Km/h. Um dos recursos disponíveis no mercado que permite realizar simulações *software-in-the-loop* é o *software* Gazebo.

Gazebo é uma ferramenta de código aberto que permite realizar simulações de robótica em 3D usando cenários realísticos. Ao utilizar o simulador é possível testar algoritmos e projetar robôs. Ele também torna possível simular com precisão a eficiência de robôs em ambientes internos e externos. A interação com o Gazebo pode ser feita utilizando *Application Programming Interfaces* (API) ou *Graphical User Interface* (GUI) (GAZEBO, 2021). A Figura 2 mostra o VANT Typhoon H480 e um mapa no *software* Gazebo.

Figura 2 – *Software* Gazebo



(a) Mapa do aroporto de San Carlos no simulador Gazebo

(b) Typhoon H480 no simulador Gazebo

Fonte: (PX4, 2020) (PX4, 2021a).

O Gazebo permite utilizar a controladora PX4 para controlar um modelo do veículo no ambiente simulado fazendo com que a interação ocorra da mesma maneira que no ambiente real, sendo que essa seria realizada através de uma API *offboard* ou um rádio controle (PX4, 2021c).

O simulador comunica com a PX4 utilizando a biblioteca MAVSDK. Essa biblioteca provém uma API simples para comunicação com sistemas MAVLink gerenciando e permitindo o acesso aos dados do veículo (DRONECODE, 2021).

MAVLink é um protocolo de mensagens leve para comunicação com VANTs e comunicação entre seus componentes. Nesse protocolo os dados são publicados em tópicos enquanto os sub-protocolos como o de missão são do tipo ponto a ponto com retransmissão (PROJECT, 2021).

Abaixo, em Código-fonte 1 tem-se o exemplo resumido na linguagem C para uma rotina

de voo. Nele é feito o uso de algumas funções da biblioteca MAVSDK. Inicialmente os motores do VANT são armados, depois ele decola pairando no ar por 10 segundos, então o veículo é pousado.

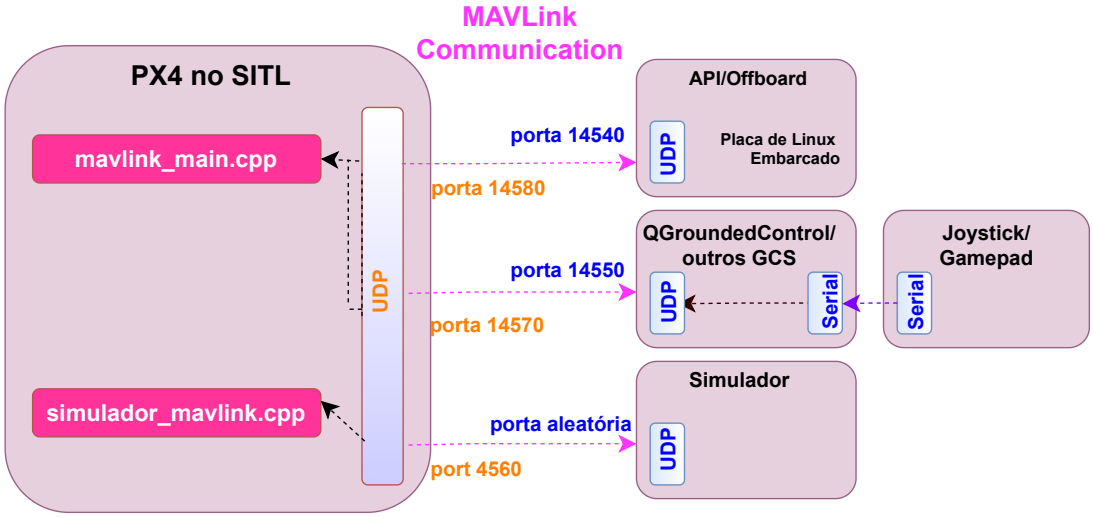
Código-fonte 1 – Exemplo de decolagem e pouso

```
1 {...}
2 const Action::Result arm_result = action.arm();
3 const Action::Result takeoff_result = action.takeoff();
4 sleep_for(seconds(10));
5 const Action::Result land_result = action.land();
6 {...}
```

Fonte: Adapatado de (JULIAN OES, 2021).

Como para este trabalho não se dispõe do *hardware* físico, será implementada a técnica *software-in-the-loop* para realização de testes práticos. No diagrama da Figura 3 é possível observar o ambiente de *software-in-the-loop*. Nessa técnica as partes do sistema se conectam via protocolo UDP e podem estar no mesmo computador ou em computadores diferentes, desde que na mesma rede. A PX4 usa um módulo específico de simulação para escutar na porta TCP 4560, o simulador conecta nessa porta para trocar informações utilizando a API MAVLink para simuladores. Para a conexão com as estações de base a PX4 usa o módulo normal da MAVLink na porta 14550. APIs desenvolvidas externamente e placas de Linux embarcado usam a porta 14540. Já a conexão serial é usada para conectar *hardwares* de *Joystick* e *Gamepad* através do QGroundControl (PX4, 2021c). Neste trabalho optou-se por utilizar uma placa de Linux embarcado Raspberry Pi 3 para executar a aplicação que integra o veículo a Blockchain.

Figura 3 – Ambiente de simulação *software-in-the-loop*



Fonte: Adapatado de (PX4, 2021c).

A comunicação entre a PX4 e a Blockchain pode ser feita com o uso da biblioteca

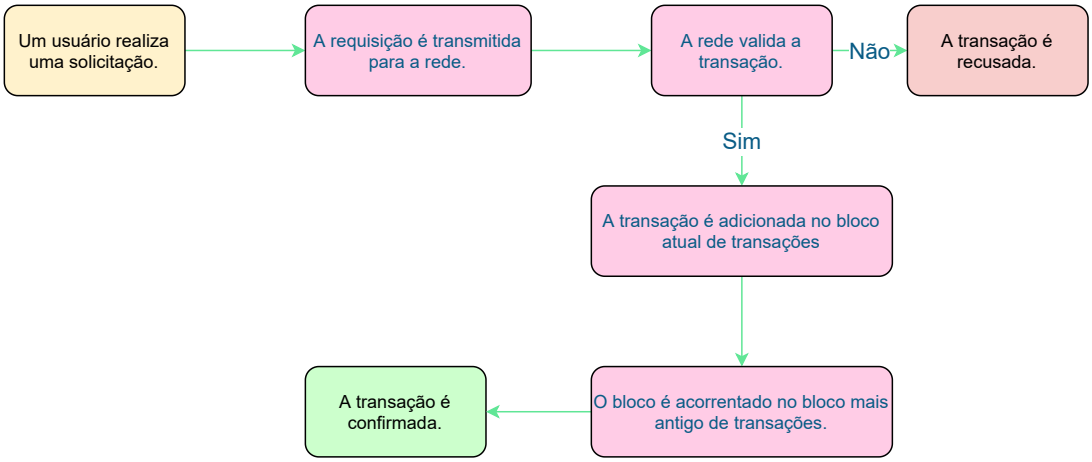
Python Web3.py. Essa API permite a interação com a Blockchain Ethereum e é muito utilizada em aplicações descentralizadas para enviar transações, interagir com o *Smart Contract* e efetuar leitura dos dados no bloco (PIPER MERRIAM, 2018). Nessa comunicação é utilizado o protocolo para chamadas remotas JSON-RPC que é um protocolo de chamada de procedimento remoto, leve e sem estado (GROUP, 2013).

Outra técnica de simulação que pode ser utilizada para testes quando não se dispõe do *hardware* físico é o *model-in-the-loop*, técnica em que a lógica de controle é simulada em laço de repetição com o modelo dinâmico do veículo, atuadores, sensores e o ambiente de simulação (GIETELINK *et al.*, 2006).

2.2 BLOCKCHAIN

Uma Blockchain é uma base de dados distribuída, em que todas as transações validadas são armazenadas em uma cadeia de blocos que cresce conforme novos blocos são acrescentados (ZHENG *et al.*, 2018). Geralmente não possui uma autoridade central, tendo seu controle mantido pelo consenso dos membros da rede distribuídos de forma descentralizada, sendo considerada uma aplicação descentralizada ou DApps. Tais membros são tidos como os nós da rede (WU *et al.*, 2021). Na Figura 4 é mostrado o funcionamento básico de uma Blockchain, que inicia a partir da solicitação por um usuário, sendo esta validada pela rede e inclusa em um bloco.

Figura 4 – Fluxo de funcionamento de uma Blockchain



Fonte: Adaptado de Laurence (2019).

Uma rede Blockchain pode ser categorizada a partir do modelo de permissão utilizado para definir quem pode interagir com a rede. Em uma rede pública ou sem permissão todos os usuários na rede podem publicar e ler. Assim, para evitar que usuários consigam efetuar

transações fraudulentas a rede utiliza um método de consenso. Já quando a rede é com permissão ou privada, é necessária uma autoridade central para gerenciar os usuários que podem publicar blocos na rede. Elas podem também restringir os usuários que podem ler os blocos da rede. Igualmente possuem um método de consenso para publicar blocos e devido ao certo nível de confiança entre os usuários por terem sua identidade conhecida, esse modelo acaba se tornando mais rápido e computacionalmente barato do que o modelo de consenso para uma rede sem permissão (YAGA *et al.*, 2019). Algumas empresas como IBM, Huawei, Oracle e Google fornecem soluções comerciais utilizando Blockchains privadas.

2.2.1 Componentes de uma Blockchain

Segundo Yaga *et al.* (2019), os principais componentes de uma Blockchain são seu *ledger*, as transações, a criptografia de chave assimétrica, a função *hash* criptográfica, os endereços, os blocos e como eles são encadeados.

2.2.1.1 Ledger

O *ledger* é um livro contábil implementado de forma distribuída utilizado para armazenar o registo de todas as transações efetuadas. Uma rede Blockchain possuí pares que podem estar espalhados por todo o mundo conectados em uma rede P2P. Cada nó da rede mantém uma cópia do *ledger* fazendo com que a rede então não possua um ponto central de ataque. Todo nó que se conecta da rede recebe uma réplica atualizada do *ledger* e mantém ela atualizada. Por ter várias cópias distribuídas a perda ou destruição da Blockchain é difícil (YAGA *et al.*, 2019; GREVE *et al.*, 2018).

2.2.1.2 Criptografia assimétrica

Para garantir a segurança do sistema, transações com integridade, autoridade, não-repudio e autenticidade, são utilizados mecanismos de *criptografia*. O método de consenso implementado também é muito importante para manter a confiabilidade do sistema (GREVE *et al.*, 2018).

A *criptografia hash* é utilizada em uma Blockchain para várias operações. Na derivação de endereço, criação de identificadores únicos, segurança do bloco de dados e segurança do

cabeçalho do bloco. Ao utilizar o método *hashhing* em um dado é possível provar que ele não foi alterado, pois é calculado uma saída única para cada conjunto de dado. Algumas características importantes das funções de *criptografia hash* são resistência à pré-imagem, sendo computacionalmente inviável calcular a entrada a partir da saída. Ela também possui resistência à segunda pré-imagem, isso significa que não é possível encontrar duas entradas com a mesma *hash* na saída (YAGA *et al.*, 2019).

2.2.1.3 Mecanismo de consenso

Quando um usuário realiza uma solicitação de uma transação em uma Blockchain essa requisição é transmitida a toda rede para ser validada. Caso satisfaça as condições de validação, ela é adicionada no bloco de transações e quando cheio, esse bloco é acrescentado à rede concluindo a transação. Caso não seja validada, a transação é descartada.

Para a validação é necessário que um conjunto de nós cheguem a um consenso em relação a um dado ou estado que está sendo compartilhado. Para isso, em sistemas distribuídos, é feito o uso de mecanismos de consenso. Esses mecanismos permitem que o consenso seja alcançado entre os nós fazendo com que o sistema de banco de dados seja replicado e assim não se tem perda de dados quando um nó falha (FERDOUS *et al.*, 2020). Conforme Ferdous *et al.* (2020) os algoritmos mais comuns são o *Proof-of-Work*, *Proof-of-Stake* e os mecanismos sem recompensa.

O algoritmo de consenso *Proof-of-Work* (PoW) evita que blocos diferentes sejam adicionados na rede contendo o mesmo dado, para adicionar um bloco na rede um nó deve resolver um desafio, o nó que resolver primeiro pode adicionar o seu bloco na cadeia de blocos, recebendo uma recompensa (NGUYEN; KIM, 2018)

Ao encontrar o valor da resposta do desafio, o nó compartilha o seu bloco e a resposta com os outros nós. Então todos os mineradores da rede que nesse momento não tenham encontrado a chave para o seu desafio param de procurar e passam a validar as transações do bloco recebido (NGUYEN; KIM, 2018).

O consenso *Proof-of-Stake* (PoS) foi proposto inicialmente no fórum Bitcointalk em 2011 (QUANTUMMECHANIC, 2011). No PoS, para um nó participar do processo de criação de um bloco, ele deve provar que possui uma certa quantidade de moedas antes. Uma parte dessas moedas ficam bloqueadas como garantia, os *stakes*, e serve como garantia para que o nó se comporte conforme os protocolos. Os mineradores que conseguirem criar blocos são

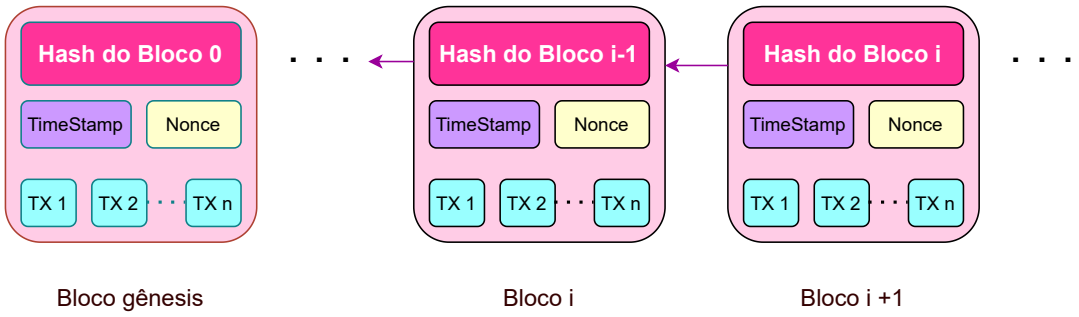
recompensados com moedas (FERDOUS *et al.*, 2020).

Já os mecanismos de consenso sem recompensa são baseados em algoritmos clássicos de consenso com alguns incrementos, dependendo do objetivo da Blockchain. São utilizados em Blockchains privadas, não relacionadas a moedas (FERDOUS *et al.*, 2020).

2.2.1.4 Blocos

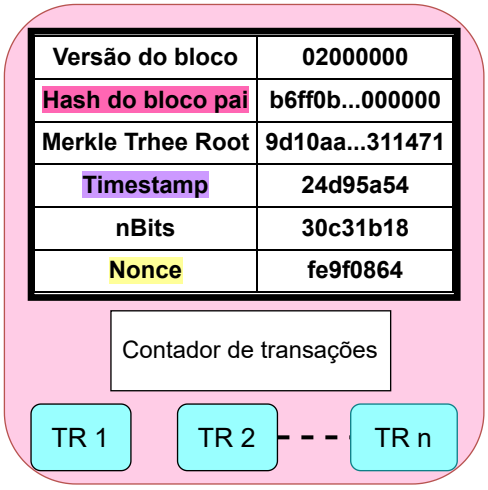
Os blocos de uma Blockchain podem ser comparados, as folhas de um livro contábil, eles armazenam os registros das transações para serem posteriormente anexados a cadeia de blocos. Cada bloco aponta diretamente para o bloco anterior por uma referência sendo essa a *hash* do bloco anterior (ZHENG *et al.*, 2018). O primeiro bloco da cadeia é chamado bloco gênesis, e como ele não tem um bloco anterior para referenciar ele recebe um valor específico, geralmente zeros (YAGA *et al.*, 2019). Na Figura 5 tem-se uma Blockchain.

Figura 5 – Blockchain



Fonte: Adapatado de (ZHENG *et al.*, 2018).

Figura 6 – Bloco de uma Blockchain



Fonte: Adapatado de (ZHENG *et al.*, 2018).

A Figura 6 mostra a estrutura detalhada de um bloco. O bloco é divido entre cabeçalho e corpo. O cabeçalho armazena a versão do bloco que define o conjunto de regras de validações a ser seguido. A *hash* do bloco pai é um valor de 256 *bits* que aponta para o bloco anterior. A *Merkle tree root hash* é a *hash* de todas as transações armazenadas no bloco. No *timestamp* é armazenado o momento em que foi realizado a transação. O nBits guarda o destino da *hash* atual compactado. Por fim é armazenado o *Nonce*, que é um valor de uso único utilizado para gerar a *hash* do bloco. No corpo do bloco são armazenadas as transações e um contador de transações (ZHENG *et al.*, 2018).

2.2.2 Smart Contract

Segundo Gonçalves (2017), um contrato é uma fonte de obrigação e para a sua formação é necessário pelo menos duas pessoas. O contrato deve atender à requisitos subjetivos, objetivos e formais para ter validade. O alto custo para substituir as leis que já são utilizadas no meio jurídico, aliado ao seu sucesso faz com que seja benéfico preservá-las e usar seus princípios quando conveniente. Novas instituições criaram maneiras de formalizar relacionamentos que só são possíveis devido à revolução digital. O uso de contratos inteligentes pode diminuir os custos legais dos negócios que fazem transações através de várias jurisdições. Também podem aumentar a privacidade enquanto diminuem as vulnerabilidades. Eles conseguem elevar a observabilidade e a verificabilidade diminuindo a dependência de leis locais e tradições (SZABO, 1996).

O termo *Smart Contract* é definido em 1994 por Nick Szabo como um protocolo de transação computadorizado que executa os termos de um contrato. O objetivo geral dos contratos inteligentes é satisfazer condições contratuais comuns diminuindo as exceções acidentais tanto quanto as maldosas e reduzindo a necessidade de um intermediário de confiança. Essas condições do contrato podem ser o pagamento, confidencialidade ou até mesmo a execução dele. Uma função importante do contrato inteligente é a capacidade de transmitir com clareza as condições definidas em seus termos (SZABO, 1994).

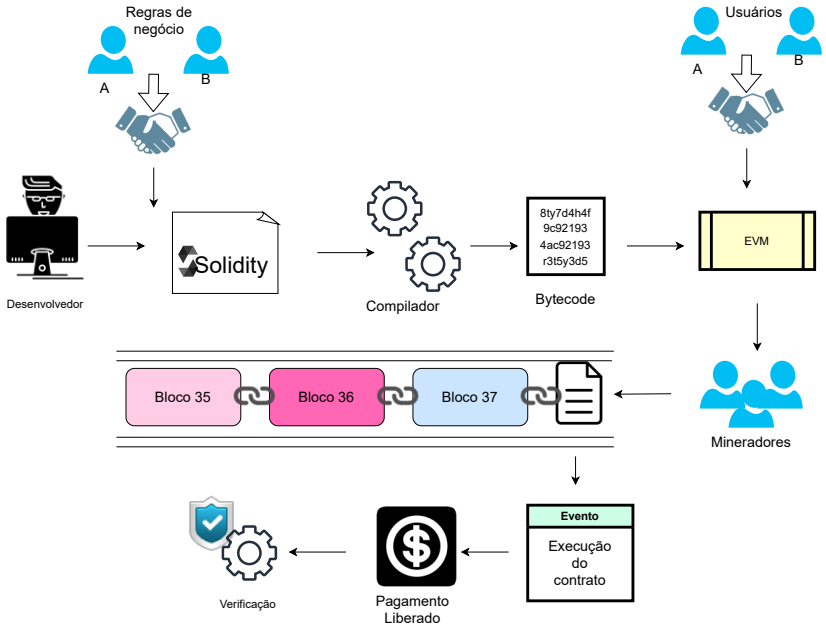
O *Smart Contract* é implementado como um conjunto de programas executados na Blockchain e recebem requisições, chamam funções pré-definidas e armazenam resultados na Blockchain (WU, 2019). Ethereum foi a primeira plataforma para desenvolvimento de contratos inteligentes em uma Blockchain, ela dispõe de uma Blockchain pública e permite escrever *Smart Contract* em Solidity. Utiliza o método de consenso (PoW) sendo que para executar transações e contratos inteligentes na Ethereum é necessário pagar os custos computacionais do processamento

em *gas*, unidade monetária extraída a partir do ether (JUNQUEIRA, 2020). Além da Ethereum estão disponíveis no mercado outras Blockchains que permitem utilizar contratos inteligentes como a Polkadot, Cardano, Solana e Binance Smartchain.

Um contrato inteligente em uma Blockchain pode ser implementado em várias linguagens de programação como Solidity, Simplicity, Vyper e Rholang. Solidity é uma linguagem de alto nível, orientada a objeto. Em Solidity um contrato inteligente é estruturado de forma semelhante às classes em programação orientada a objeto. O código do contrato contém variáveis e funções que leem e alteram os valores das variáveis (ETHEREUM, 2021). O contrato escrito é compilado em *bytecode* e então implantado na Blockchain da Ethereum (WU, 2019).

A Ethereum possui um tipo de usuário para aceitar códigos compilados, um usuário do contrato com os mesmos recursos do usuário individual, mas é controlado pelo *Smart Contract*. Transações para usuários de contratos serão convertidas para requisições para o contrato, sendo que cada nó possui um espaço livre reservado para armazenar os dados resultantes da execução do contrato (WU, 2019).

Figura 7 – Ciclo de um *Smart Contract* na Blockchain Ethereum



Fonte: Adapatado de (SAYEED; MARCO-GISBERT; CAIRA, 2020).

Na Figura 7 observa-se a sequência de execução de uma *Smart Contract* na Blockchain Ethereum. Um desenvolvedor escreve um contrato em Solidity baseado nas regras de negócios entre as partes. O código é compilado em *Bytecode* e executado na *Ethereum Virtual Machine* (EVM). Os usuários realizam uma transação no *Smart Contract*. Os mineradores incluem o contrato na Blockchain e este será executado em data definida quando ele foi escrito. Ao ser

executado, o pagamento é liberado (SAYEED; MARCO-GISBERT; CAIRA, 2020). É necessária uma verificação após o pagamento ser liberado, pois, não é possível saber com certeza quando a transação será executada devido ao custo de *gas* da Blockchain.

Código-fonte 2 – Exemplo de *Smart Contract* em Solidity

```
1 pragma solidity ^0.4.0;
2 contract ArmazenamentoSimples{
3     uint guardaDado;
4     function set(uint x) {
5         guardaDado = x;}
6     function get() constant returns (uint) {
7         return guardaDado;}
8 }
```

Fonte: Adapatado de (ETHEREUM, 2017).

Em Código-fonte 2 é mostrado um exemplo de *Smart Contract* nomeado como *ArmazenamentoSimples* e implementado em *Solidity*. Esse contrato permite alterar e ler o valor da variável *guardaDado* utilizando as funções *set* e *get* respectivamente.

3 MATERIAIS E MÉTODO

Neste capítulo serão apresentados os materiais utilizados no desenvolvimento do sistema, a proposta inicial de arquitetura e a descrição das etapas do desenvolvimento.

3.1 MATERIAIS

O desenvolvimento e a simulação de testes para o sistema proposto foi realizado em uma máquina com o sistema operacional Debian, o ambiente de desenvolvimento Visual Studio Code, o ambiente de simulação Gazebo, o emulador de Blockchain Ganache CLI e a controladora PX4.

As aplicações foram desenvolvidas na linguagem Python e o *Smart Contract* em Solidity, com as bibliotecas Python Web3 para interação com a Blockchain, MAVSDK para interação com a PX4 e o gerenciador de versões do compilador solc para Solidity, o py-solc-x.

Quadro 2 – Especificações das licenças

Ferramenta	**Licença**	**Versão**
Debian	GNU	11 (bullseye)
Gazebo	Apache 2.0	11.1.0
Ganache CLI	MIT	v7.0.3
MAVSDK	BSD 3-clause	1.2.0
Web3	MIT	8.3.1
PX4 Flight Stack e Middleware	BSD	1.8.11
Pixhawk Hardware	CC-BY-SA 3.0	
py-solc-x	MIT	1.1.1

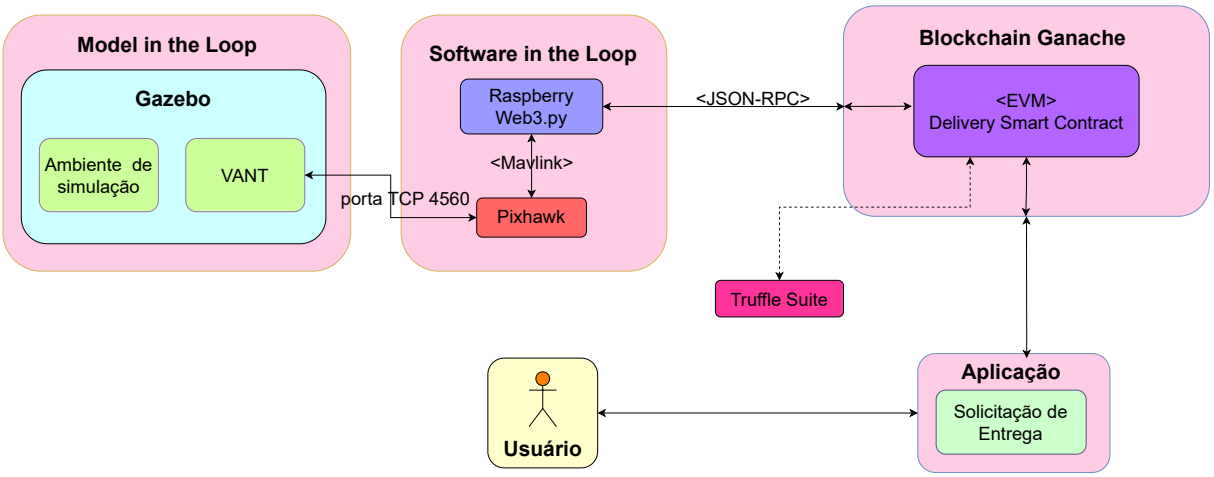
Fonte: Autoria própria (2022).

Foi optado pelo uso de ferramentas *open source* como é possível observar na tabela de especificações de licenças no Quadro 2.

3.2 MÉTODO

Seguindo a arquitetura proposta na Figura 8 foram desenvolvidas 3 aplicações Pyhton, uma para permitir a interação do veículo na simulação Gazebo com a Blockchain, uma para implantação do *Smart Contract* na Ethereum e outra para solicitação de entrega. As partes de interface de usuário não foram contempladas nesse trabalho.

Figura 8 – Arquitetura prosposta para o sistema



Fonte: Autoria própria (2022).

O *Smart Contract* foi escrito na linguagem de programação Solidity dentro de uma aplicação Python, compilado com a ferramenta solc-x e implantado na Blockchain pela biblioteca Web3 e a Blockchain Ethereum foi emulada pelo *software* Ganache CLI.

A Raspberry Pi foi simulada por uma aplicação Python recorrendo à coleção de bibliotecas MAVSDK para interagir com a PX4. Foi aplicada à técnica *software-in-the-loop* para a simulação da PX4 que controla o VANT Typhoon emulado no simulador de sistemas dinâmicos Gazebo. Foi acrescentado um ambiente a simulação permitindo assim utilizar coordenadas reais como destino. O ambiente de simulação e o modelo do veículo são simulados com a técnica *Model in the Loop*. Para a comunicação entre a Ethereum e a aplicação Python utilizou-se a biblioteca Web3. A aplicação que solicita as entregas também foi desenvolvida em Python com a biblioteca Web3 para ler e escrever dados no *Smart Contract* de solicitação de entrega.

A validação foi feita a partir da simulação de algumas condições:

- Cadastro de veículo a ser utilizado na entrega.
- Solicitação de entrega para o totem um, dois e três.
- Solicitação de entrega com pagamento menor e maior do que o custo de entrega.
- Solicitação de entrega para um veículo indisponível.
- Resgate do valor pertencente a outra carteira.

Os códigos e configurações estão disponíveis no repositório GitHub que pode ser acessado em https://github.com/Leticia147/Delivery.

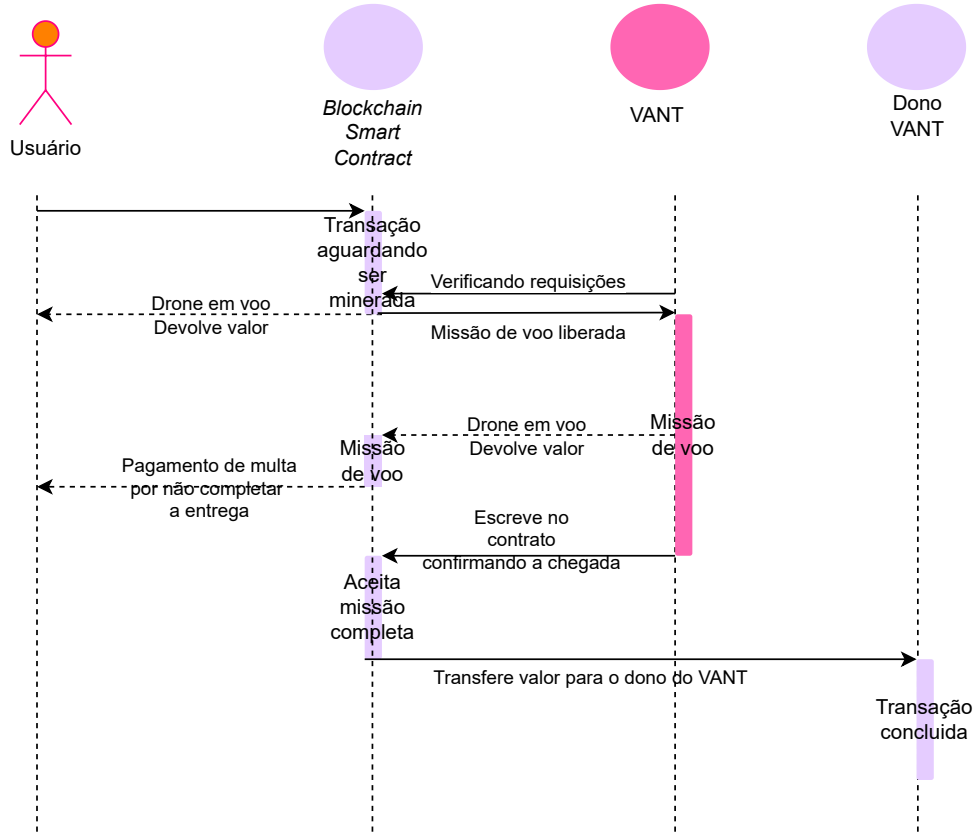
4 SISTEMA DE ENTREGA COM VEÍCULOS AÉREOS NÃO TRIPULADOS

Este capítulo apresentará detalhes do sistema desenvolvido a partir da arquitetura proposta. Esse sistema permite a solicitação de entregas utilizando VANTs e *Smart Contract*, abrangendo desde o cadastro do veículo até a transferência de pagamento para o dono ao completar a entrega.

4.1 IMPLEMENTAÇÃO

O diagrama sequencial da Figura 9 mostra o fluxo para uma solicitação de entrega que inicia a partir do pedido de um usuário ao *Smart Contract* e se essa transação de solicitação for aceita é acrescentada como um bloco na Blockchain.

Figura 9 – Diagrama sequencial para solicitação de entrega



Fonte: Autoria própria (2021).

O veículo realiza leituras periódicas no *Smart Contract* verificando a existência de solicitações pendentes e ao verificar que existe uma solicitação de entrega ele inicia uma missão de voo até o ponto de destino da solicitação. Ao chegar no destino, o VANT escreve no contrato

que a missão foi finalizada, permitindo a transferência do valor referente ao pagamento da entrega para o dono.

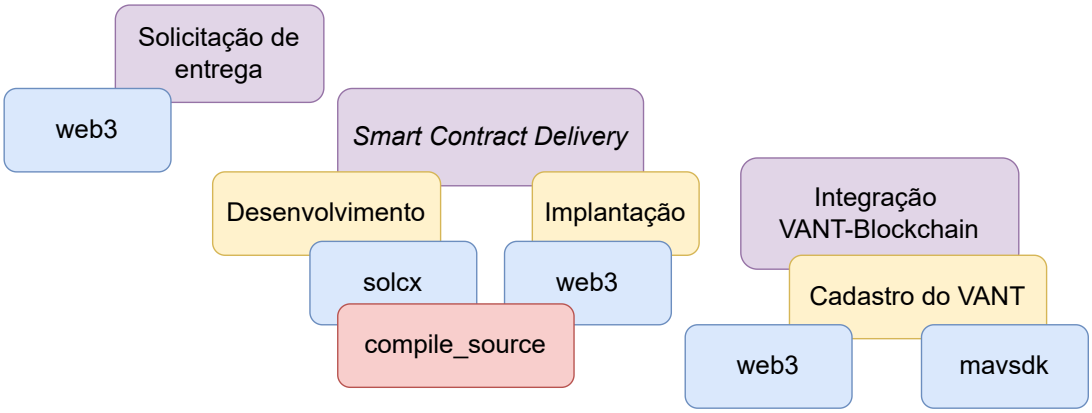
Caso no momento de validação de uma solicitação de entrega seja verificado que o valor enviado é menor que 10 Ether a transação não é aceita, caso o valor seja maior é devolvido o troco. Se a solicitação de entrega for para um veículo ocupado ela também não é aceita e o valor é estornado.

4.1.1 Arquitetura do Sistema

A arquitetura desenvolvida e implementada neste trabalho está disponível na Figura 8. Ela contempla a simulação do veículo, o *Smart Contract* e a aplicação que permite ao usuário solicitar entregas.

Para atender a arquitetura foram desenvolvidas 3 aplicações Python conforme é apresentado na Figura 10. Foi implementado uma aplicação para permitir a solicitação de entrega pelo usuário, outra para o desenvolvimento e implantação do *Smart Contract* na Blockchain e uma para a integração do VANT com a Blockchain.

Figura 10 – Aplicações Python desenvolvidas



Fonte: Autoria própria (2022).

As simulações de entregas são feitas no *software* Gazebo, com a PX4 sendo executada em *software-in-the-loop*, assim é possível testar o comportamento da controladora de voo de maneira muito próxima à controladora com *hardware* físico em um ambiente real. Também é acrescentado a simulação o mapa da região de San Carlos Airport, sendo possível utilizar as coordenadas condizentes com pontos reais de GPS como destino das entregas. A Blockchain Ethereum é emulada localmente pelo *software* Ganache CLI, permitindo o *deploy* do *Smart Contract* escrito em Solidity.

4.1.2 Emulação Local da Blockchain Ethereum

A Blockchain Ethereum foi simulada localmente utilizando a ferramenta Ganache CLI, sendo a versão em linha de comando do *software* Ganache. Na Listing 4.1 é mostrado a execução da ferramenta ao digitar o comando Ganache no terminal de comandos.

Listing 4.1 – Ganache CLI

```
> ganache
ganache v7.0.3 (@ganache/cli: 0.1.4, @ganache/core: 0.1.4)
Starting RPC server
Available Accounts
==================
(0) 0x091371E1E4375B2C103ded1f32DE5236E50E3469 (1000 ETH)
...
Private Keys
==================
(0) 0x56387115ed51131a157a[...]bc7b4c06e2e521ffa7cc58c5e1
...
HD Wallet
==================
Mnemonic: reason sunny attract plunge [...] frost afford
Base HD Path:  m/44'/60'/0'/0/{account_index}
Default Gas Price
==================
2000000000
BlockGas Limit
==================
30000000
Call Gas Limit
==================
50000000
Chain Id
==================
1337
RPC Listening on 127.0.0.1:8545
```

Fonte: Autoria própria (2022).

A ferramenta exibe os dados referente as 10 contas pré-configuradas e disponibilizadas, cada uma contém 1000 unidades de Ether com validade somente para a Ethereum local e geradas

a cada simulação. Também são exibidas as configurações da Blockchain com os custos para as transações. Utilizando as configurações padrão do Ganache é possível interagir com a Blockchain na rede local porta RPC 8545 e implantar o *Smart Contract*.

4.1.3 Implementação do Smart Contract

As funções do *Smart Contract* foram desenvolvidas a partir das regras de negócio mostradas na listagem abaixo:

- Os veículos aéreos utilizados para realizar as entregas são do modelo Typhoon H480.
- O endereço de carteira utilizado no cadastro do veículo é também o endereço que vai receber os pagamentos referentes a esse VANT.
- As entregas são feitas entre 3 pontos de entrega pré-definidos, chamados aqui neste trabalho de totem.
- O custo para cada entrega é fixo em 10 Ether.
- Um usuário poderá solicitar a entrega somente de um totem para outro.
- Após solicitar uma entrega ela não poderá ser cancelada.
- Não é permitido solicitar uma entrega para um veículo que já está em missão de entrega.

O *Smart Contract* foi escrito na linguagem de programação Solidity sendo executado na Blockchain. Neste trabalho ele é responsável pela validação das solicitações de entregas, transferir valores e permitir o cadastro de veículos. Foi utilizado uma aplicação Python para seu desenvolvimento e implantação na Ethereum local.

Com a função *compile_source* da biblioteca *solcx*, a aplicação Python pode compilar o código-fonte do *Smart Contract Delivery* escrito em Solidity. Após a compilação, a função retorna a ABI e o EVM *bytecode* do contrato. O *Application Binary Interface* (ABI) possui o escopo das funções do *Smart Contract Delivery* e o tipo dos argumentos, definindo como as funções do contrato devem ser chamadas. Já o EVM *bytecode* é o código compilado do *Smart Contract*, podendo ser executado na *Ethereum Virtual Machine* (EVM).

Usando a biblioteca *Web3* a aplicação conecta a Ethereum local, após a conexão é possível setar um endereço como conta padrão, foi escolhido arbitrariamente a conta zero entre as fornecidas pela Ethereum. O *bin* e o ABI são passados como parâmetros para a função *contract* da *Web3* para criar o objeto do contrato sendo então submetida a transação que implanta esse contrato. Ao minerar o bloco referente a implantação do *SmartContract* o Ganache exibe o *log*

da transação, conforme Listing 4.2.

Listing 4.2 – Bloco do *SmartContract*

```
Transaction: 0x31ef0c51dc90543f4241[...]393250f8eff9dd7c0acb
Contract created: 0xe76352c2b51e59d0c4a708b9949a664d4328d08f
Gas usage: 1302301
Block number: 1
Block time: Sun May 29 2022 15:46:23 GMT-0300
```

Fonte: Autoria própria (2022).

Em Código-fonte 3 é listado resumidamente o *Smart Contract Delivery* com suas funções. No início do código é definido a versão de Solidity em que o *smart contract* é escrito, evitando quebra de compatibilidade entre versões da linguagem, seguindo pelo contrato nomeado como *Delivery*.

Inicialmente no código do *Smart Contract Delivery* foi definido a *struct Drone* que armazena os dados do veículo, o endereço de carteira do seu dono, se o VANT está em missão de voo e as coordenadas do destino da entrega. Após foi definido um *mapping* para armazenar os veículos cadastrados e na sequência foram escritas as funções que permitem a interação com *Smart Contract*.

A função *registerDrone* instancia um objeto do tipo *Drone* com os dados passados na chamada da função e o acrescenta a estrutura *registerDrones*. A cada chamada o valor da variável de controle da quantidade de veículos cadastrados é incrementada.

As funções *setDestinyOne*, *setDestinyTwo* e *setDestinyThree* têm o mesmo comportamento e a finalidade de solicitar entregas. Quando chamada, a função muda o estado do veículo para em missão e seta suas coordenadas de destino. Ao chamar a função, o usuário precisa escolher o veículo desejado e transferir a taxa de entrega de 10 Ether, esse valor fica armazenado no contrato até a entrega ser finalizada. Se o valor enviado for maior é devolvido o troco.

Ao completar a entrega, o veículo chama a função *inTheDestiny*, ela define o estado do veículo como disponível para que o *Smart Contract* transfira o valor referente a uma entrega ao dono do VANT. Toda vez que ocorre uma chamada bem sucedida de escrita para uma função do *Smart Contract*, seja alterando o valor de alguma variável ou adicionando um novo objeto *Drone*, um bloco referente a transação é acrescentado a Blockchain.

Código-fonte 3 – *Smart Contract Delivery*

```
pragma solidity >0.5.0;

contract Delivery {
    uint qntDrone;
    struct Drone {
        droneOwner;
        uint id_drone;
        uint flying;
        string latitude_deg;
        string longitude_deg;
    }
    mapping (uint => Drone) public registerDrones;

    function registerDrone() public returns (uint) {
        //Permite o cadastro de veículos.
    }

    function inTheDestiny(uint id_drone) public {
        //Indica que o veículo chegou ao destino.
    }

    function setDestinyOne(uint id_drone) public payable returns (bool){
        //Define o primeiro totem como destino da entrega.
    }

    function setDestinyTwo(uint id_drone) public payable returns (bool){
        //Define o segundo totem como destino da entrega.
    }

    function setDestinyThree(uint id_drone) public payable returns (bool){
        //Define o terceiro totem como destino da entrega.
    }

    function getStatusDrone(uint id_drone) public view returns (string
   memory,string  memory,uint) {
        //Retona o estado atual do veículo.
    }
}
```

Fonte: Autoria própria (2022).

4.1.4 Simulação do Veículo no Gazebo

Para a simulação de entregas foi utilizado o *software* Gazebo que por padrão possui um ambiente de simulação vazio. Esse ambiente foi substituído pelo mapa da região do aeroporto de San Carlos Airport, localizado na cidade de San Carlos na Califórnia, Estado Unidos, ambiente fornecido pelo Gazebo. Também foi acrescentado à simulação o modelo de veículo Typhoon H480 mantido pela PX4.

É possível iniciar a simulação da PX4, o Gazebo com o mapa e o VANT Typhoon a partir do diretório em que a PX4 foi instalada, executando o comando:

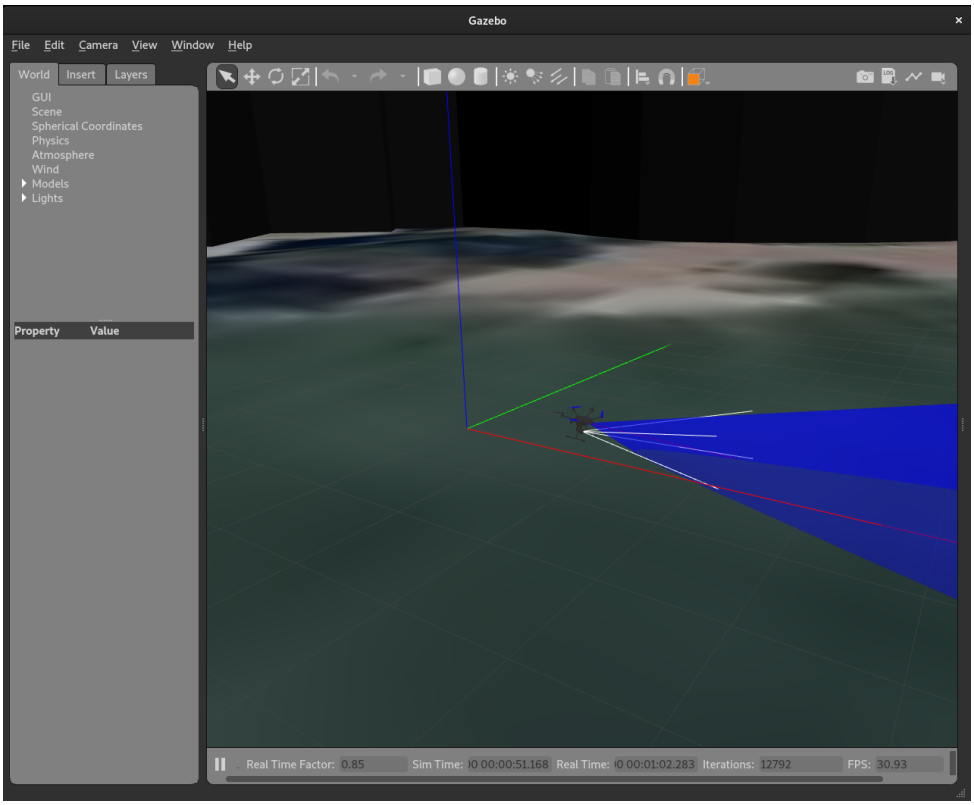
Listing 4.3 – Execução do Gazebo e SITL

```
$ make px4_sitl gazebo_typhoon_h480__ksql_airport
```

Fonte: Autoria própria (2022).

Ao iniciar a simulação com o comando *make* é utilizado as configurações padrão do Gazebo, sendo o simulador e o SITL lançados no mesmo computador e as portas configuradas automaticamente. A porta TCP 4560 do simulador é usada para comunicação com a PX4. A PX4 escuta nessa porta enquanto o simulador transmite dados nessa porta esperando para começar a comunicação. A PX4 utiliza a *Simulator MAVLink API* para trocar informações com o Gazebo, podendo ler os dados dos sensores do veículo na simulação e enviar comando e valores para os atuadores do veículo na simulação. A Figura 11 mostra a simulação no ambiente no Gazebo.

Figura 11 – Ambiente de simulação Gazebo



Fonte: Autoria própria (2022).

4.1.5 Integração VANT-Blockchain

Como esse trabalho não abrange o *hardware* físico, a aplicação Python que permite a comunicação entre a controladora Pixhawk e a Blockchain Ethereum, executada na arquitetura pela Raspbery, é executada nas simulações pela IDE.

Nesta aplicação foi instanciado um objeto do tipo System da biblioteca MAVSDK. Esse objeto é utilizado para a comunicação com a PX4 que controla o veículo simulado no Gazebo. Para realizar a conexão é passado para o método *connect* desse objeto o endereço e porta que a PX4 está sendo emulada. Também foi instanciado um objeto do tipo Web3 da biblioteca Web3 e nomeado como w3, ele foi conectado ao endereço e porta do Ganache e possibilita realizar transações na Blockchain. É possível ler e escrever no *Smart Contract* passando como parâmetro para o objeto w3 o ABI e o endereço de sua implantação na Ethereum e instanciando um objeto *contract*.

É chamado a função *registerDrone* do *Smart Contract* para cadastrar o veículo que está sendo simulado no Gazebo. Em um *loop*, a cada 5 segundos o objeto w3 chama a função *getStatusDrone* do *Smart Contract* para verificar se existem solicitações para o veículo cadastrado e as coordenadas de destino.

Quando existirem solicitações é utilizado o modo missão de voo da PX4 para controlar o veículo até o destino da entrega, quando o *upload* da missão para a PX4 é concluído, a conexão com a aplicação para essa entrega não é mais necessária. Após pousar, é chamado a função *inTheDestiny* do *Smart Contract* informando que a entrega foi finalizada, recebendo o pagamento.

4.1.6 Solicitação de Entrega

A aplicação que permite a solicitação de entrega foi desenvolvida em Python. Ela utiliza a biblioteca Web3 para conexão com a Blockchain e interação com o *Smart Contract*.

Código-fonte 4 – Solicitação de entrega

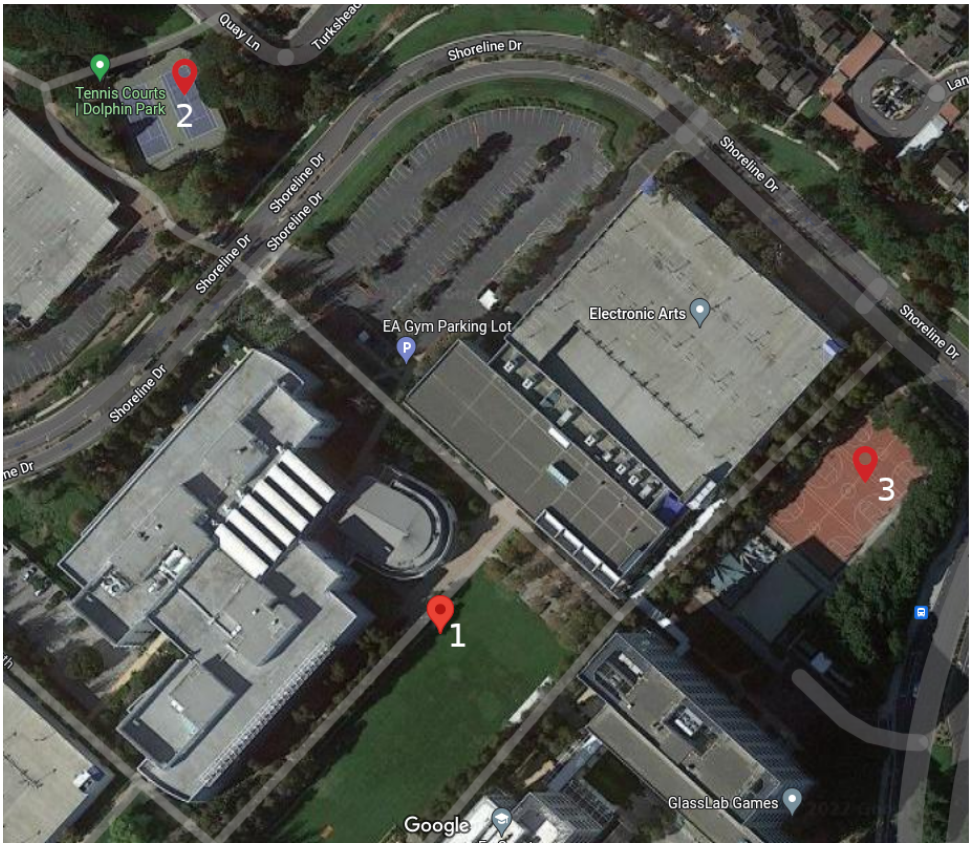
```
1  from web3 import Web3
2  w3 = Web3(Web3.HTTPProvider('http://127.0.0.1:8545'))
3  w3.eth.default_account = w3.eth.accounts[2]
4
5  abi = [{'inputs': [], 'stateMutability' ... }]
6  contract_id = Web3.toChecksumAddress('0x4fb3cc0d ... 92c93412')
7
8  entrega = w3.eth.contract(address = contract_id,abi = abi)
9
10 entrega.functions.setDestinoTwo(1).transact({'from': w3.eth.default_account
       ,'value': w3.toWei(10,'ether')})
11 }
```

Fonte: Autoria própria (2022).

Conforme é mostrado no Código-fonte 4, a aplicação utiliza o ABI e o endereço do contrato implantado na Ethereum para chamar a função do *Smart Contract* que permite a

solicitação de entregas, passado como parâmetro o número do veículo a ser utilizado.

Figura 12 – Mapa com os destinos de entrega



Fonte: Google Maps (2022).

Os pontos de destino escolhidos estão ilustradas na Figura 12. O primeiro totem encontra-se no gramado próxima a EA Sports, o totem dois está na quadra de tênis da Tennis Courts, e o terceiro ponto de entrega disponível é a quadra de basquete próxima a Eletronic Arts.

4.2 RESULTADOS

Neste capitulo são abordados e discutidos os testes realizados para validação da arquitetura proposta para o sistema. São verificados os comportamentos do veículo e do *Smart Contract* nos casos descritos na Seção 3.2. Para validar os casos de uso, foram executados o Gazebo conforme Listing 4.3 e o Ganache CLI conforme Listing 4.1.

4.2.1 Caso de Uso 1

O primeiro caso de uso validado foi o comportamento do sistema para o cadastro de um veículo utilizando a aplicação *Pyhton* da Seção 4.1.5. A aplicação comunica com a *Ethereum*

passando o endereço da carteira da conta de número 0 e chamando a função *registerDrone* do *Smart Contract*. Ao finalizar a transação é acrescentado o bloco referente ao cadastro do veículo a Blockchain, conforme mostrado na Listing 4.4.

Listing 4.4 – Bloco referente ao cadastro de veículo

```
Transaction: 0x8e1660a393286863df1e[...]53ff3d84f72b9f12f2d9d
Gas usage: 116028
Block number: 2
Block time: Sun May 29 2022 16:42:40 GMT-0300
```

Fonte: Autoria própria (2022).

4.2.2 Caso de Uso 2

Para a validação de solicitação de entrega, uma aplicação *Python* usando o endereço de carteira número dois, efetua um pedido de entrega para o veículo no totem um e escolhe como destino o totem dois e envia o pagamento exato para uma entrega. Para isso é chamada a função *setDestinyTwo* do *Smart Contract Delivery* passando como parâmetro o número do veículo um e transferindo o valor de 10 Ether como pagamento. Um bloco referente a transação de solicitação é acrescentado a Blockchain.

Ao finalizar a entrega o veículo chama a função *inTheDestiny* do *Smart Contract* ficando disponível para novas missões e recebendo o pagamento referente a entrega. Um bloco com os dados da transação é acrescentado a Blockchain.

Para o primeiro e o terceiro totem apenas a função chamada no *Smart Contract* difere, nesse caso é chamado a função *setDestinyOne* e a *setDestinyThree* respectivamente.

4.2.3 Caso de Uso 3

Para observar o comportamento do sistema referente a uma solicitação de entrega com pagamento menor que 10 Ether foi efetuado um pedido de entrega partindo do totem um como destino o totem dois e enviado o valor de 5 Ether como pagamento. Quando a função *setDestinyTwo* é chamada ela verifica que o valor é menor que 10 retornando a exceção mostrada na Listing 4.5, invalidando a transação.

Listing 4.5 – Exceção para valor menor

```
Ocorreu uma exceção: ContractLogicError
```

```
execution reverted: VM Exception while processing transaction: revert
    O valor deve ser maior que 10 Ether
```

Fonte: Autoria própria (2022).

Por ser uma transação não validada pelo *Smart Contract* não é acrescentado um bloco referente a ela a Blockchain.

4.2.4 Caso de Uso 4

Para a validação de pagamento com valor maior que 10 Ether, foi efetuado uma solicitação de entrega enviando como pagamento o valor de 15 Ether. Ao ser chamada, a função do *Smart Contract* verificou que o valor é maior, devolvendo o troco para a carteira que solicitou a entrega. Nesse caso não é retornado nenhuma exceção, seguindo o fluxo normal de uma solicitação de entrega com um bloco referente a transação adicionado a Blockchain.

4.2.5 Caso de Uso 5

Inicialmente foi solicitada uma entrega para o veículo, com ele em missão foi chamada a função *setDestinyTwo*. O *Smart Contract* verificou que o veículo não estava disponível, retornando a exceção mostrada na Listing 4.6.

Listing 4.6 – Exceção para veículo indisponível

```
Ocorreu uma exceção: ContractLogicError
execution reverted: VM Exception while processing transaction:
    revert Veiculo nao esta disponivel.
```

Fonte: Autoria própria (2022).

Essa exceção indica que o veículo não pode aceitar novas solicitações de entregas enquanto estiver completando uma missão.

4.2.6 Caso de Uso 6

Nesse caso foi verificado o comportamento da função *inTheDestiny* para uma tentativa de resgate de valor não pertencente a carteira que está efetuando a chamada da função. Para isso foi solicitado uma entrega e com o veículo em missão foi chamada a função e passado o endereço de carteira diferente do dono do veículo. A função do *Smart Contract* retornou a exceção que

está na Listing 4.7.

O valor referente ao pagamento da entrega é transferido para a carteira do dono do VANT quando o veículo chama a função *inTheDestiny* informando que finalizou a entrega. Essa função valida se o endereço de quem está chamando a função é igual ao que foi utilizado para realizar o cadastro do VANT.

Listing 4.7 – Exceção para não dono do veículo

```
Ocorreu uma exceção: ContractLogicError
execution reverted: VM Exception while processing transaction:
    revert Voce nao e o dono do veiculo!
```

Fonte: Autoria própria (2022).

Caso o dono do veículo chame a função quando não está finalizando uma missão e não tem valor a receber, é retornado a exceção da Listing 4.8.

Listing 4.8 – Exceção para veículo no destino

```
Ocorreu uma exceção: ContractLogicError
execution reverted: VM Exception while processing transaction:
    revert Veiculo ja esta no destino!
```

Fonte: Autoria própria (2022).

Se na chamada da função for passado como parâmetro o número de um veículo não cadastrado, é retornado a exceção da Listing 4.9.

Listing 4.9 – Exceção para veículo não cadastrado

```
Ocorreu uma exceção: ContractLogicError
execution reverted: VM Exception while processing transaction:
    revert Veiculo nao esta cadastrado
```

Fonte: Autoria própria (2022).

As exceções são retornadas pelo *Smart Contract*.

4.2.7 Caso de Uso 7

Também foi verificado o estado de um veículo cadastrado. Isso é possível a partir da chamada da função *getStatusDrone* pois ela retorna os dados de latitude e longitude do destino da última solicitação de entrega e se o veículo está em missão de entrega referente ao VANT passado como parâmetro. O retorno do exemplo abaixo é referente a um veículo em missão de entrega para o destino dois.

Listing 4.10 – Estado do veículo

```
['37.52520345925217', '-122.2561141299747', 1]
```

Fonte: Autoria própria (2022).

As coordenadas retornadas pela função são referente ao ponto de entrega número dois e o valor 1 representa que o veículo está ocupado em uma missão de entrega.

5 CONCLUSÃO

Neste trabalho foi desenvolvido um sistema de entregas utilizando veículo aéreo não tripulado e Blockchain. Para isso, foram elaboradas três aplicações Python, uma para a implantação do *Smart Contract* na Blockchain, outra para integrar o veículo e a Blockchain permitindo a leitura e escrita no *Smart Contract* e a terceira para solicitação de entrega.

O sistema foi desenvolvido utilizando a Blockchain Ethereum com o modelo de veículo aéreo Typhoon H480 e o mapa da região próxima ao aeroporto de San Carlos simulados pelo ambiente Gazebo com a controladora PX4 executada em *software-in-the-loop*. Para integração entre as partes foram utilizadas as bibliotecas MAVSDK, Web3 e o gerenciador de compiladores solc para Python, py-solc-x.

Devido ao custo do hardware físico não foi possível realizar testes físicos, mas com o simulador Gazebo e a técnica *software-in-the-loop* foi possível simular e validar o funcionamento do sistema para os casos de teste. Por utilizar veículos aéreos não tripulados a distância das entregas e tamanhos dos pacotes ficam restritos a categoria de veículos utilizado. Também devido ao custo da moeda Ether não foi possível realizar a implantação do *Smart Contract* na Ethereum, mas com o uso Ganache CLI foi possível testar o seu comportamento em uma Blockchain Ethereum executada localmente. Não foi desenvolvido nesse trabalho um tratamento para caso o VANT não consiga completar a missão de entrega como uma multa a ser paga pelo dono do veículo, por exemplo.

Foi possível atingir os objetivos propostos neste trabalho visto que o sistema permite a solicitação de entregas a partir da chamada de funções de um *Smart Contract* implantado em uma Blockchain com as entregas realizadas por um veículo aéreo não tripulado controlado em um ambiente de simulação. O controle da rota até o destino é feito pela controladora PX4 com o modo missão de voo.

Para trabalhos futuros, propõe-se a implementação da arquitetura disponível em Apêndice A, essa arquitetura abrange o desenvolvimento de uma interface para o cadastro de veículos e solicitação de entregas. Também propõe-se o desenvolvimento de um modelo de multas para quando o veículo não concluir a entrega.

REFERÊNCIAS

AGUIAR, G. **Esta startup apostou no delivery por drone e já tem o iFood como cliente**. [*S. l.: s. n.*], ago. 2021. Exame. Disponível em: https://exame.com/negocios/speedbird-e-a-startup-brasileira-que-tornou-realidade-delivery-com-drone/. Acesso em: 9 abr. 2021.

ALAIMO, A. *et al.* Mathematical modeling and control of a hexacopter. In: IEEE. 2013 International Conference on Unmanned Aircraft Systems (ICUAS). [*S. l.: s. n.*], 2013. P. 1043–1050.

ANAC. **Instrução Suplementar - IS: IS n21-002 Revisão A.[S.1.]** [*S. l.: s. n.*], 2012. P. 21.

ARANTES, J. d. S. **Sistema autônomo para suspervisão de missão e segurança de voo em VANT's**. Jul. 2019. F. 217. Tese (Doutorado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

AYED, M. B.; ZOUARI, L.; ABID, M. Software in the loop simulation for robot Manipulators. **Engineering, Technology & Applied Science Research**, v. 7, n. 5, 2017.

BEKMEZCI, I.; SAHINGOZ, O. K.; TEMEL, Ş. Flying ad-hoc networks (FANETs): A survey. **Ad Hoc Networks**, Elsevier, v. 11, n. 3, p. 1254–1270, 2013.

CANADA, D. D. **Reimagining the Way You Deliver**. [*S. l.: s. n.*], 2021. Disponível em: https://dronedeliverycanada.com/. Acesso em: 14 nov. 2021.

CHANG, S. E.; CHEN, Y.-C.; LU, M.-F. Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process. **Technological Forecasting and Social Change**, Elsevier, v. 144, p. 1–11, 2019.

CHANG, Y. S.; LEE, H. J. Optimal delivery routing with wider drone-delivery areas along a shorter truck-route. **Expert Systems with Applications**, Elsevier, v. 104, p. 307–317, 2018.

CHEUNG, K.-F.; BELL, M. G. Attacker–defender model against quantal response adversaries for cyber security in logistics management: An introductory study. **European Journal of Operational Research**, Elsevier, 2019.

CLAESSON, A. *et al.* Time to delivery of an automated external defibrillator using a drone for simulated out-of-hospital cardiac arrests vs emergency medical services. **Jama**, American Medical Association, v. 317, n. 22, p. 2332–2334, 2017.

COLLOTTA, M.; PAU, G.; CAPONETTO, R. A real-time system based on a neural network model to control hexacopter trajectories. In: IEEE. 2014 International Symposium on Power Electronics, Electrical Drives, Automation and Motion. [*S. l.: s. n.*], 2014. P. 222–227.

DRONECODE. **MAVSDK (main)**. [*S. l.: s. n.*], set. 2021. Disponível em: https://mavsdk.mavlink.io/main/en/index.html. Acesso em: 15 nov. 2021.

ETHEREUM. **Introdução aos Smart Contracts**. [*S. l.: s. n.*], 2017. Disponível em: https://solidity-portuguese.readthedocs.io/pt/latest/introduction-to-smart-contracts.html. Acesso em: 30 nov. 2021.

ETHEREUM. **Solidity**. [*S. l.: s. n.*], 2021. Disponível em: https://docs.soliditylang.org/en/v0.8.10/. Acesso em: 10 nov. 2021.

FERDOUS, M. S. *et al.* Blockchain consensus algorithms: A survey. **arXiv preprint arXiv:2001.07091**, 2020.

FERRAG, M. A.; MAGLARAS, L. DeliveryCoin: An IDS and blockchain-based delivery framework for drone-delivered services. **Computers**, Multidisciplinary Digital Publishing Institute, v. 8, n. 3, p. 58, 2019.

FLYTREX. **FAQ**. [*S. l.: s. n.*], 2021. Disponível em: https://www.flytrex.com/faq. Acesso em: 11 nov. 2021.

FOGAÇA, A. **Anac autoriza testes para entregas via drones**. [*S. l.: s. n.*], ago. 2021. Tecnoblog. Disponível em: https://tecnoblog.net/358597/anac-autoriza-testes-para-entregas-via-drones/. Acesso em: 2 nov. 2021.

FOGAÇA, A. **iFood testa entregas por drones em Campinas**. [*S. l.: s. n.*], ago. 2021. Tecnoblog. Disponível em: https://tecnoblog.net/358647/ifood-testa-entregas-por-drones-em-campinas/. Acesso em: 2 nov. 2021.

GAZEBO. **Gazebo Simulation**. [*S. l.: s. n.*], 2021. GitBook. Disponível em: https://dev.px4.io/v1.10_noredirect/en/simulation/gazebo.html. Acesso em: 6 nov. 2021.

GIETELINK, O. *et al.* Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations. **Vehicle System Dynamics**, Taylor & Francis, v. 44, n. 7, p. 569–590, 2006.

GONÇALVES, C. R. **Direito civil brasileiro 3-contratos e atos unilaterais**. [*S. l.*]: Saraiva Educação SA, 2017.

GREVE, F. G. *et al.* Blockchain e a Revolução do Consenso sob Demanda. **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)-Minicursos**, 2018.

GROUP, J.-R. W. **JSON-RPC 2.0 Specification**. [*S. l.: s. n.*], jan. 2013. Disponível em: https://www.jsonrpc.org/specification. Acesso em: 30 nov. 2021.

HASAN, H. R.; SALAH, K. Blockchain-based proof of delivery of physical assets with single and multiple transporters. **IEEE Access**, IEEE, v. 6, p. 46781–46793, 2018.

HIMANSHU JOSHI, S. M. **Target Drone Market Statistics 2021-2030:** [*s. l.: s. n.*], out. 2021. Disponível em: https://www.alliedmarketresearch.com/target-drone-market-A06235. Acesso em: 18 nov. 2021.

ISERMANN, R.; SCHAFFNIT, J.; SINSEL, S. Hardware-in-the-loop simulation for the design and testing of engine-control systems. **Control Engineering Practice**, Elsevier, v. 7, n. 5, p. 643–653, 1999.

JOERSS, M.; NEUHAUS, F.; SCHRÖDER, J. How customer demands are reshaping last-mile delivery. **The McKinsey Quarterly**, v. 17, p. 1–5, 2016.

JULIAN OES, M. G. **Example: Takeoff and Land**. [*S. l.: s. n.*], 2021. Disponível em: https://github.com/mavlink/MAVSDK/blob/main/examples/takeoff_and_land/takeoff_and_land.cpp. Acesso em: 1 dez. 2021.

JUNQUEIRA, N. R. **Concessão de Permissão a Dados de Saúde Baseada em Contratos Inteligentes em Plataforma de Blockchain**. Mar. 2020. F. 89. Diss. (Mestrado) – Instituto de Informática, Universidade Federal de Goiás, Goiânia.

KEENEY, T. **Amazon Drones Could Deliver a Package in Under Thirty Minutes for Less Than One Dollar**. [*S. l.: s. n.*], dez. 2015. Disponível em: https://ark-invest.com/articles/analyst-research/amazon-drone-delivery/. Acesso em: 11 nov. 2021.

KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IEEE. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566). [*S. l.: s. n.*], 2004. v. 3, p. 2149–2154.

KOETSIER, J. **Novo google delivery: drones vão de loja ao cliente em minutos**. [*S. l.: s. n.*], out. 2021. Forbes Brasil. Disponível em: https://forbes.com.br/forbes-tech/2021/10/google-lanca-servico-de-delivery-com-uso-drones/. Acesso em: 2 nov. 2021.

LAURENCE, T. **Blockchain for dummies**. [*S. l.*]: John Wiley & Sons, 2019.

LEE, H. L. *et al.* Technological disruption and innovation in last-mile delivery. **Value Chain Innovation Initiative**, 2016.

LING, G.; DRAGHIC, N. Aerial drones for blood delivery. **Transfusion**, Wiley Online Library, v. 59, S2, p. 1608–1611, 2019.

MEOLA, D.; IANNELLI, L.; GLIELMO, L. Flight control system for small-size unmanned aerial vehicles: Design and software-in-the-loop validation. In: IEEE. 21ST Mediterranean Conference on Control and Automation. [*S. l.: s. n.*], 2013. P. 357–362.

NETO, M. P. d. M. **Veículos Aéreos Não Tripulados e Sistema de Entrega: Estudo, Desenvolvimento e Testes**. Fev. 2016. F. 103. Diss. (Mestrado) – UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE, Natal.

NGUYEN, G.-T.; KIM, K. A survey about consensus algorithms used in blockchain. **Journal of Information processing systems**, Korea Information Processing Society, v. 14, n. 1, p. 101–128, 2018.

OTTO, A. *et al.* Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. **Networks**, Wiley Online Library, v. 72, n. 4, p. 411–458, 2018.

PIPER MERRIAM, J. C. **Introduction**. [*S. l.: s. n.*], 2018. Disponível em: https://web3py.readthedocs.io/en/stable/. Acesso em: 30 nov. 2021.

PROJECT, D. **MAVLink Developer Guide**. [*S. l.: s. n.*], 2021. GitBook. Disponível em: https://mavlink.io/en/. Acesso em: 15 nov. 2021.

PX4. **Gazebo Vehicles**. [*S. l.: s. n.*], jun. 2021. Disponível em: https://docs.px4.io/master/en/simulation/gazebo_vehicles.html#typhoon_h480. Acesso em: 15 nov. 2021.

PX4. **Gazebo Worlds**. [*S. l.: s. n.*], nov. 2020. Disponível em: https://docs.px4.io/master/en/simulation/gazebo_worlds.html. Acesso em: 15 nov. 2021.

PX4. **PX4 Autopilot User Guide**. [*S. l.: s. n.*], set. 2021. Disponível em: https://docs.px4.io/master/en/. Acesso em: 15 nov. 2021.

PX4. **Simulation**. [*S. l.: s. n.*], 2021. Dronecode. Disponível em: https://dev.px4.io/v1.9.0_noredirect/en/simulation/. Acesso em: 20 nov. 2021.

QUANTUMMECHANIC. **Proof of stake instead of proof of work**. [*S. l.: s. n.*], jul. 2011. Bitcointalk. Disponível em: https://bitcointalk.org/index.php?topic=27787.0. Acesso em: 8 nov. 2021.

SAYEED, S.; MARCO-GISBERT, H.; CAIRA, T. Smart contract: Attacks and protections. **IEEE Access**, IEEE, v. 8, p. 24416–24427, 2020.

SHAKHATREH, H. *et al.* Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges. **Ieee Access**, IEEE, v. 7, p. 48572–48634, 2019.

STOLAROFF, J. K. *et al.* Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery. **Nature communications**, Nature Publishing Group, v. 9, n. 1, p. 1–13, 2018.

STRAWN, G. BLOCKCHAIN. **IT Professional**, v. 21, n. 1, p. 91–92, 2019. DOI: 10.1109/MITP.2018.2879244.

SZABO, N. **Smart contracts**. [*S. l.: s. n.*], 1994.

SZABO, N. Smart contracts: building blocks for digital markets. **EXTROPY: The Journal of Transhumanist Thought,(16)**, v. 18, n. 2, 1996.

VERMA, V.; LEGER, C. SSim: NASA Mars rover robotics flight software simulation. In: IEEE. 2019 IEEE Aerospace Conference. [*S. l.: s. n.*], 2019. P. 1–11.

WANG, Q.; SU, M. Integrating blockchain technology into the energy sector—from theory of blockchain to research and application of energy blockchain. **Computer Science Review**, Elsevier, v. 37, p. 100275, 2020.

WU, K. An empirical study of blockchain-based decentralized applications. **arXiv preprint arXiv:1902.04969**, 2019.

WU, K. *et al.* A first look at blockchain-based decentralized applications. **Software: Practice and Experience**, Wiley Online Library, v. 51, n. 10, p. 2033–2050, 2021.

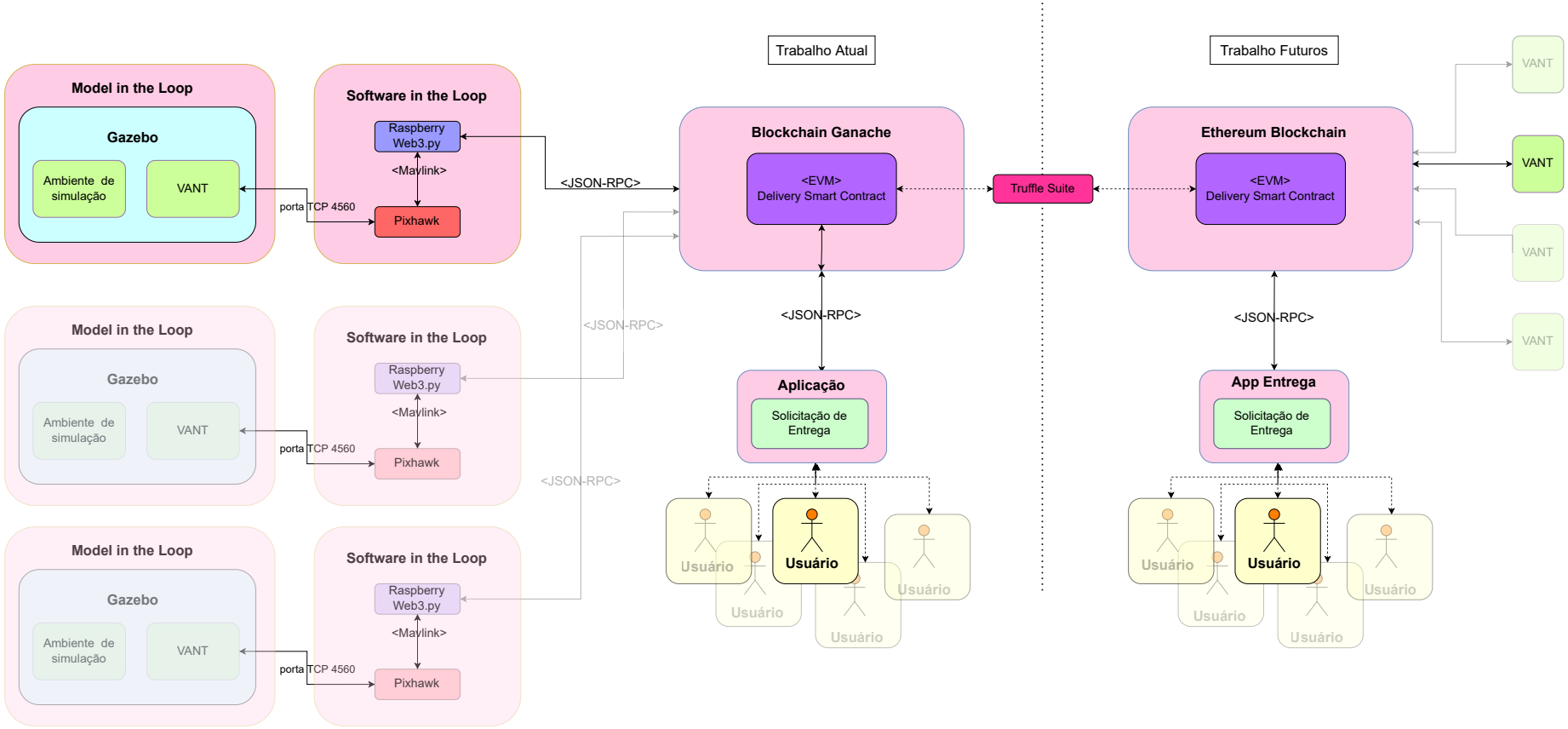
YAGA, D. *et al.* Blockchain technology overview. **arXiv preprint arXiv:1906.11078**, 2019.

YOO, W.; YU, E.; JUNG, J. Drone delivery: Factors affecting the public's attitude and intention to adopt. **Telematics and Informatics**, Elsevier, v. 35, n. 6, p. 1687–1700, 2018.

ZHENG, Z. *et al.* Blockchain challenges and opportunities: A survey. **International Journal of Web and Grid Services**, Inderscience Publishers (IEL), v. 14, n. 4, p. 352–375, 2018.

APÊNDICE A — ARQUITETURA COMPLETA DO SISTEMA

Figura 13 – Arquitetura completa para o sistema de entrega



Fonte: Autoria Própria (2022).