DESENVOLVIMENTO DE UM *DRONE* AUTONÔMO GUIADO POR MEIO DE TÉCNICAS DE VISÃO COMPUTACIONAL

Vítor Albiero*
Herculano Haymussi De Biasi**

RESUMO

No presente artigo apresenta-se o processo de desenvolvimento de um *drone* com capacidade de se mover de forma autônoma, utilizando-se para isso, técnicas de visão computacional. O projeto é baseado na plataforma Arduino e no microcomputador Raspberry Pi, utilizando-se o sistema operacional Linux, assim como a linguagem de programação C++ e a biblioteca de visão computacional *open source* OpenCV. Assim como a visão computacional (conjunto de técnicas que procura fazer com que máquinas "enxerguem"), os *drones*, inicialmente desenvolvidos com propostas militares, estão ganhando cada vez mais espaço em diversas outras áreas. Com isso, mostra-se necessário um sistema inteligente para que se consiga operá-los de forma autônoma, fazendo com que sejam utilizados com maior eficiência e sem a necessidade de controle humano, automatizando, assim, as áreas nas quais são utilizados. No sistema autônomo, o *drone* voa e realiza operações por intermédio do processamento dos vídeos capturados por uma câmera, passando as instruções de direção e sentido para o computador, o qual direciona o *drone* para o local desejado. Ao final do projeto, é apresentado um *drone* autônomo capaz de executar instruções por meio do reconhecimento de um objeto colorido em um ambiente controlado.
Palavras-chave: Visão computacional. Drone. Arduino. Raspberry Pi. OpenCV.

1 INTRODUÇÃO

Ganhando cada vez mais espaço, a visão computacional é uma das áreas mais promissoras da computação. Com o intuito de fazer os computadores "enxergarem", ela possibilita tarefas antes não possíveis, como conduzir um veículo de forma autônoma, reconhecer pessoas em vídeos, ajudar médicos a determinar doenças, entre outros.

Inicialmente projetados para uso militar, os veículos aéreos não tripulados (VANTs), mais conhecidos como *drones*, estão ganhando espaço em diversas aplicações civis em que seria inviável um ser humano trabalhar, realizando tarefas arriscadas e servindo como verdadeiras ferramentas de trabalho. Dentro das inúmeras atividades que um *drone* pode desenvolver, destaca-se o seu uso em filmagens áreas, mapeamento de florestas, transportes industriais, entregas de mercadorias, fiscalização de fronteira e vigilância.

Hoje, a utilização dos *drones* fica condicionada ao controle manual de um operador, que comanda o aparelho a longas distâncias por meio de um rádio controle, sendo, por causa disso, suscetível a falhas de operação e imprecisões de posicionamento. Atualmente, os VANTs conseguem navegar sozinhos utilizando o GPS (Sistema Global de Posição, do inglês *Global Positioning System*), porém, ele por si só não traz uma grande precisão. Essa limitação motivou o desenvolvimento deste projeto: desenvolver um *drone* autônomo que seja capaz de se orientar sozinho por intermédio do processamento visual.

Com a utilização do processamento de imagens, é possível que o *drone* reconheça formas como setas e objetos. Uma vez reconhecidas tais formas, ou objetos, estas podem funcionar como instruções, semelhantes às placas de trânsito, fazendo com que o veículo aéreo não tripulado possa realizar as funções programadas para elas. O resultado deste trabalho é um produto que pode ser utilizado em diversas áreas, possibilitando automatizar processos, como transporte de peças em ambientes industriais, filmagens aéreas automáticas, navegação em ambientes pequenos, entre outros.

* Egresso do Curso de Ciência da Computação da Universidade do Oeste de Santa Catarina de Videira; vitor_albiero@outlook.com
** Professor do Curso de Ciência da Computação da Universidade do Oeste de Santa Catarina de Videira; herculano.debiasi@unoesc.edu.br

2 **FUNDAMENTAÇÃO TEÓRICA**

Esta seção fornece uma breve introdução às tecnologias, componentes e ferramentas utilizados para o desenvolvimento desse projeto.

2.1 LINGUAGEM DE PROGRAMAÇÃO C++

O C++ é uma linguagem de programação que foi desenvolvida para estender a linguagem C. De acordo com Stroustrup (2013), o C++ é uma linguagem de programação com uso geral que proporciona um eficiente e direto modelo de compatibilidade com *hardware* combinado com facilidades no desenvolvimento de soluções leves.

Pela sua capacidade de alto e baixo nível, e por aceitar diversas técnicas de programação e suas inúmeras funções, o C++ é uma das linguagens mais utilizadas em *softwares* de dispositivos embarcados, servidores e *firmwares*, permitindo que diversas implementações possam ser realizadas com bom desempenho.

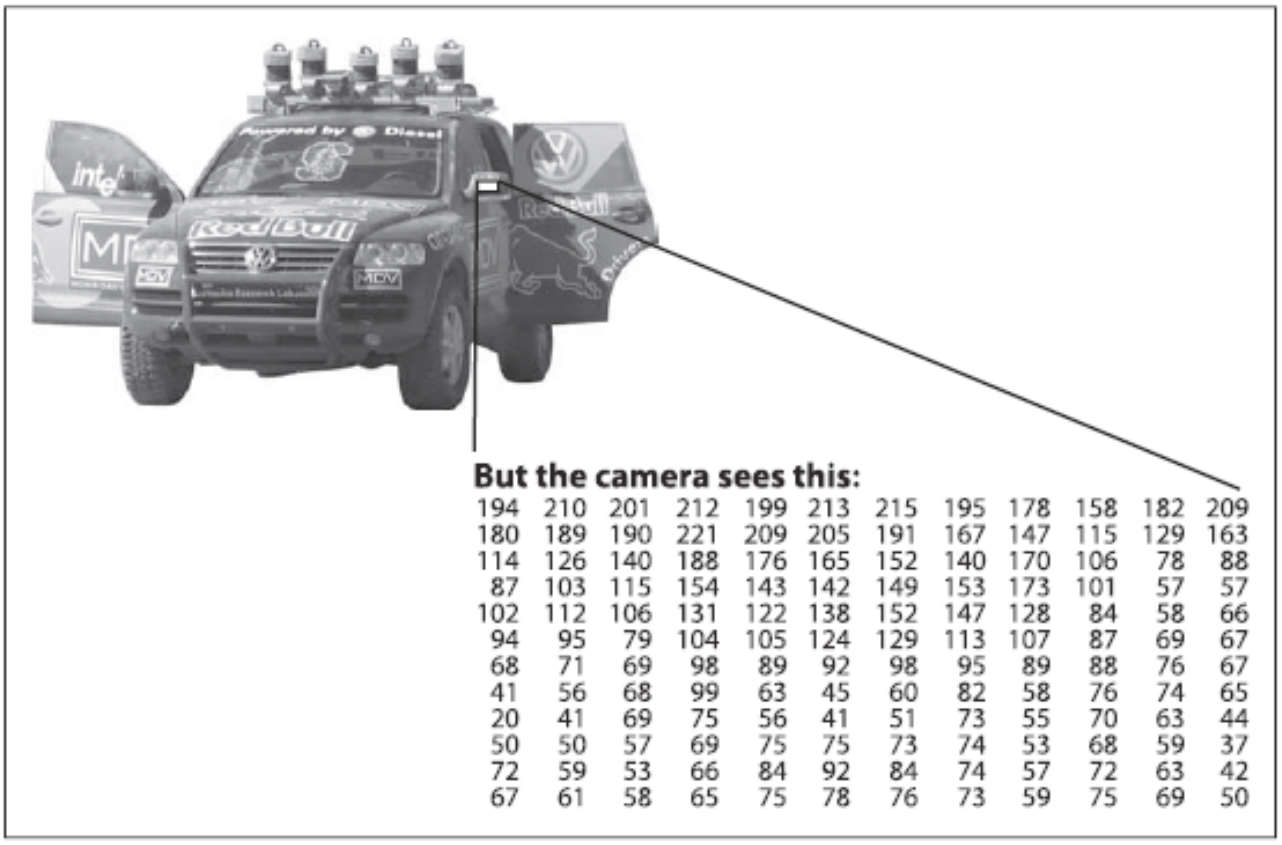
2.2 VISÃO COMPUTACIONAL

A visão computacional é a área da computação responsável por fazer o computador "enxergar" o mundo, possibilitando, assim, que o computador realize funções com base no que está sendo "visto". A visão computacional pode ser definida como a transformação de dados de uma foto ou vídeo para uma decisão ou representação computacional, isto é, um algoritmo, em vista de se atingir um objetivo ou realizar alguma tarefa (BRADSKI; KAEHLER, 2008).

De acordo com Bradski e Kaehler (2008), em um sistema de visão computacional, ao receber uma imagem ou vídeo, o computador enxerga na verdade uma matriz com números da câmera ou imagem, e somente isto.

A Imagem 1 mostra a foto de um veículo, e o que o computador está "enxergando" do espelho é somente uma matriz de números, com uma grande distorção de valores. A tarefa da visão computacional é, então, transformar essa matriz de valores distorcidos em um padrão ou representação e formar a percepção.

Imagem 1 – Como um computador enxerga uma imagem



Fonte: Bradski e Kaehler (2008).

A visão computacional é uma das áreas que mais cresce no ramo da computação e pode parecer simples, pois as pessoas são extremamente baseadas no visual. Porém o computador interpretar tais dados e transformá-los em algoritmos é uma tarefa um tanto quanto complicada, sendo simplificada com a utilização de bibliotecas, como a OpenCV.

2.3 OPENCV

A OpenCV (*Open Source Computer Vision*, que em português significa Visão Computacional de Código Aberto) é uma biblioteca de programação que possui funções de processamento de imagens, objetivando realizar tarefas de visão computacional em tempo real.

A biblioteca é escrita em C/C++ otimizado e possui interfaces adicionais para Python e Java; suporta os sistemas operacionais Linux, Windows e Mac OS X. A OpenCV foi desenvolvida com foco em aplicações de tempo real, tirando proveito do processamento multinúcleo (DALAL, 2013).

A maior intenção da OpenCV é fornecer uma infraestrutura simples de visão computacional para o desenvolvimento de aplicações visuais sofisticadas de maneira rápida e eficiente, contendo mais de quinhentas funções que podem ser utilizadas nas diversas áreas, incluindo inspeção de qualidade de produtos em linhas de produção, segurança, interface de usuário e robótica. Outra característica da OpenCV é a sub-biblioteca MLL *(Machine Learnig Library* – Biblioteca de Aprendizado de Máquina), que tem como foco armazenar algo previamente reconhecido e identificar tais elementos em tarefas subsequentes (BRADSKI; KAEHLER, 2008).

A OpenCV é distribuída sob a licença BSD (*Berkeley Software Distribution*), isto quer dizer que aplicações comerciais podem usá-la sem a necessidade de liberar o código fonte. Entretanto, existem algoritmos completos dentro da OpenCV que possuem patentes, restringindo ou limitando o uso deles (DALAL, 2013).

2.4 PLATAFORMA ARDUINO

A plataforma Arduino é um ambiente de prototipagem eletrônica *open hardware* que utiliza um microcontrolador para processar os dados recebidos e enviados nas portas de entrada e saída, assim como possui uma linguagem de programação que é baseada em C/C++. O Arduino é a plataforma de desenvolvimento, que integra *hardware* e *software*, que mais cresce atualmente, pois é de fácil implementação, baixo custo e de alta produtividade.

O Arduino é um ambiente de desenvolvimento composto por dois elementos: *hardware* e *software*. O *hardware* consiste em uma placa eletrônica com um microcontrolador, normalmente da família ATMega, que faz a leitura das entradas e envia os sinais de saída. Esse microcontrolador pode ser utilizado na placa que o Arduino é vendido, ou pode ser separado e montado em uma nova placa, conforme necessário. O microcontrolador é composto por entradas e saídas digitais e analógicas; as analógicas possuem variação de valores numéricos, diferente das digitais que trabalham somente com 0 e 1. O segundo elemento, *software*, é o ambiente de desenvolvimento integrado Arduino (IDE) *open source*, baseado em C/C++, que é responsável pela programação (BANZI, 2008).

Para o desenvolvimento de um projeto Arduino é necessário realizar as conexões físicas entre os componentes e realizar a programação na IDE conforme o desejado (MCROBERTS, 2011).

2.4.1 **Arduino Crius**

O modelo do Arduino escolhido para este projeto é o CRIUS AIO Pro v2. Este modelo foi desenvolvido especialmente para ser utilizado como controlador de voo, possuindo sensores e conexões próprias para a ligação dos demais componentes. Ele possui um microcontrolador ATMega 2560 com 256 Kbps de memória, um acelerômetro, giroscópio, barômetro e magnetômetro. Com isso, não é necessário adquirir componentes extras para que o *drone* possa voar de forma estabilizada e com controle de altura.

2.4.2 **Arduino Uno**

O Arduino UNO é um dos modelos mais utilizados da plataforma Arduino, por ser de baixo custo e tamanho reduzido. Seu microcontrolador também pode ser facilmente trocado (nas versões não SMD) caso venha a queimar.

O Arduino Uno possui um microcontrolador ATMega328 com memória de 32KB, possui seis portas analógicas, quatorze portas digitais e duas portas de comunicação I2C. Possui também um regulador de tensão que trabalha com faixas de 6 a 20 volts, convertendo-as para uma tensão estável de 5 volts.

2.5 RASPBERRY PI

O Raspberry Pi é um microcomputador de baixo custo do tamanho de um cartão de crédito que roda em plataforma Linux e oferece conexões de entrada e saída de dados. O Raspberry Pi é composto, no seu modelo B, por um processador 32 bits de 700 MHz de frequência, igual ao utilizado no iPhone 3G, e possui 512 MB de memória RAM; seu armazenamento é feito por meio de um cartão SD (digitalmente seguro). Possui também duas portas USB, Ethernet, HDMI, vídeo composto, áudio analógico e porta de alimentação micro USB (RICHARDSON; WALLACE, 2013).

Além de todas as conexões supracitadas, o Raspberry Pi oferece ainda diversas portas GPIO (*General Purpose Input and Output* – Entradas e Saídas de Propósito Geral), que têm como função realizar conexões com sensores externos e, até mesmo, outras plataformas, conexão DSI (*Display Serial Interface* – interface serial de *display*) para conectar telas LCD e, também, conexão CSI (*Camera Serial Interface* – Conexão Serial de Câmera) (RICHARDSON; WALLACE, 2013).

Como o sistema utilizado no Raspberry Pi é o Linux, existem diversas distribuições disponíveis; a mais comumente utilizada é a Raspbian (nome derivado de Raspberry mais Debian), que é baseado na distribuição Debian do Linux (SCHMIDT, 2012).

Combinando o poder do sistema operacional Linux a todas as conexões suportadas, o Raspberry Pi é um computador que pode ser utilizado para projetos em que seja necessário interagir com sensores e que requeiram um poder de processamento maior do que os microcontroladores como o Arduino conseguem oferecer.

2.6 COMUNICAÇÃO I^2C

O I^2C ou IIC é a sigla para Circuito Inter-Integrado (*Inter-Integrated Circuit)*. Ele trabalha com barramento serial multimaster, de duas vias para uma comunicação eficiente entre dispositivos físicos, possibilitando, assim, enviar e receber dados de ambos os lados da conexão.

Funciona em modo *half-duplex* e utiliza a tecnologia *open-drain,* que necessita de duas vias, sendo elas a *data serial* (SDA) e o *clock serial* (SCL) conectados à alimentação por meio de resistores. Operando com endereços de 7 bits com 16 endereços reservados, o barramento I^2C consegue teoricamente interligar até 112 dispositivos. O I^2C opera em quatro velocidades diferentes: 100 Kbit/s no modo normal, 400 Kbit/s no modo veloz, 1 Mbit/s no modo mais veloz (*fast-mode plus*) e 3.4 Mbit/s no modo ultra veloz (*high-speed*) (KUGELSTADT, 2009).

2.7 VEÍCULOS AÉREOS NÃO TRIPULADOS

Os veículos aéreos não tripulados (VANTs), mais conhecidos como *drone*s, são veículos que voam sem a necessidade de pessoas a bordo para serem guiados. Foram inicialmente desenvolvidos com fins militares, porém, atualmente, exercem funcionalidades em diversas áreas civis.

Os *drones* são projetados para "[...] realizar tarefas arriscadas ao ser humano ou ferramentas para trabalhos que ninguém quer realizar" (GARRET, 2013, p. 1), sendo utilizados para vários fins, como fiscalizar fronteiras, combater criminosos, usos militares e, até mesmo, auxiliar nas vendas de imóveis por meio de filmagens aéreas (TAUFER, 2013).

Os VANTs são classificados em três tipos: os de uso militar, que são, geralmente, em formato de aviões; os *drones* de filmagens, que são multirrotores, isto é, possuem duas ou mais hélices para voar, sendo que os quadricópteros

(helicóptero de quatro hélices) são os mais comumente utilizados; e, os de brinquedo que são semelhantes aos de filmagens, porém têm menor alcance e pouca velocidade (HAMANN, 2013).

Assim, os multirrotores são perfeitos para atividades em que a estabilidade de voo é o foco, com a possibilidade de parar no ar, controlar com precisão a velocidade de movimento, realizar curvas fechadas, entre outras performances.

3 **METODOLOGIA**

O projeto foi desenvolvido em três estágios principais: o desenvolvimento do algoritmo de reconhecimento visual, o desenvolvimento do algoritmo de controle dos modos de voo e a construção física do *drone*.

3.1 SOFTWARE OPEN SOURCE MULTIWII

O MultiWii é um software *open source* desenvolvido para a plataforma Arduino, que tem como finalidade realizar o controle de voo de *drone*s multirrotores. Ele consegue suportar multirrotores de até oito hélices (MULTIWII, 2014).

O controle de baixo nível, como rotação dos motores, estabilização do *drone*, movimentos verticais e horizontais são todos controlados por meio do MultiWii mediante comandos enviados por um rádio transmissor ou computador.

3.2 ALGORITMO DE RECONHECIMENTO VISUAL

O desenvolvimento do algoritmo de reconhecimento visual teve como objetivo capturar a imagem da câmera, aplicar filtros por cor e realizar transformações morfológicas, determinando, assim, as coordenadas *x* e *y* do objeto. Por último, essas coordenadas tiveram que ser transferidas para o Arduino Uno.

O filtro por cor faz a conversão do sistema de cor RGB (*Red*, *Green*, *Blue*) para o sistema HSV (*Hue*, *Saturation*, *Value*). O sistema HSV trabalha com as informações de tonalidade, saturação e valor. O filtro por cor tem a função de fazer com que a câmera capte apenas o objeto que estiver no intervalo dos elementos HSV definidos; por exemplo, se for determinado que o intervalo da cor azul for entre 0 e 256 (valor máximo) e o intervalo das cores vermelha e verde for 0 a 0, o algoritmo "enxergará" somente objetos totalmente azuis.

Após filtrado o objeto com base em sua cor, ainda restam alguns "ruídos" de outras cores na imagem. Para eliminá-los, são aplicadas diversas transformações morfológicas na imagem. "Enxergando" apenas o objeto desejado, o algoritmo é capaz de determinar o local em que ele se encontra e transferir as informações das coordenadas *x* e *y* para o Arduino.

A Imagem 2 mostra o trecho de código responsável por determinar os pontos *x* e *y* após realizadas as operações citadas.

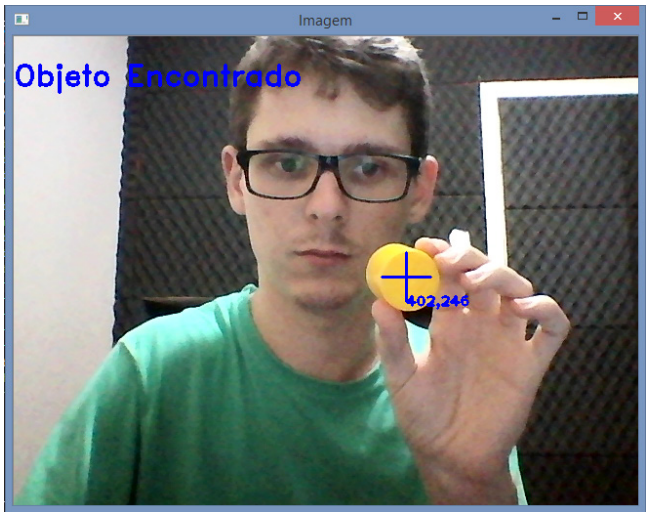
Imagem 2 – Parte do código de reconhecimento visual que encontra o objeto

```
if (matriz.size() > 0) { //verifica se tem algo aparecendo na imagem
    int qtdObjects = matriz.size(); //recupera a quantidade de objetos encontrados

    if(qtdObjects<MAX_QTD_OBJECTS) { //verifica se existem mais objetos que o limite
        for (int index = 0; index >= 0; index = matriz[index][0]) {
            Moments moment = moments((cv::Mat)contornos[index]); //encontra o objeto
            double area = moment.m00;

            if(area > MIN_OBJECT_AREA && area < MAX_OBJECT_AREA && area > refArea){
                x = moment.m10 / area; //definine as coordenadas em que o objeto se encontra
                y = moment.m01 / area;
                sendI2C(x, y); //envia os dados para o arduino uno
                objectFound = true;
                gpio(0, 0, 1); //liga um led verde indicando que o objeto foi encontrado
            } else {
                sendI2C(0,0);  //envia os dados para o arduino uno
                objectFound = false;
                gpio(0, 1, 0);  //liga um led vermelho indicando que não foi encontrado o objeto
            }
        }
    }
}
```

Fonte: os autores.

Com o intuito de validar se o algoritmo realmente está encontrando as posições corretas do objeto, o programa desenha automaticamente uma cruz e insere as coordenadas *x* e *y* na imagem da câmera, conforme mostra a Imagem 3.

Imagem 3 – Objeto encontrado e pontos marcados na visualização da câmera



Fonte: os autores.

3.3 ALGORITMO DE CONTROLE DOS MODOS DE VOO

O desenvolvimento do algoritmo no Arduino tem como base três modos de voo: modo manual, modo com sensores, e modo autônomo. A troca dos modos é feita atrás do rádio transmissor. Nesse momento, o Arduino verifica a posição em que o botão se encontra para determinar o modo de voo.

A Imagem 4 mostra a parte do código responsável pelo controle manual, no qual o Arduino Uno lê os dados que o controle está enviando e somente os repassa para o Arduino Crius.

Imagem 4 – Código responsável pelo modo de voo manual

```
aleo_in=pulseIn(5,HIGH,20000); // le dados receptor
eleo_in=pulseIn(6,HIGH,20000);
rudd_in=pulseIn(7,HIGH,20000);
thro_in=pulseIn(8,HIGH,20000);
aux1=pulseIn(9,HIGH,20000);

if((aux1<=1000) && (aux1 > 0)){ //modo manual
  aleo.writeMicroseconds(aleo_in); //manda dados para o drone
  eleo.writeMicroseconds(eleo_in);
  thro.writeMicroseconds(thro_in);
  rudd.writeMicroseconds(rudd_in);
}
```

Fonte: os autores.

O modo de voo sensorial utiliza quatro sensores ultrassônicos, que são responsáveis por dizer aonde o *drone* deve ir. Caso encontre um objeto, ele se move na direção contrária a ele. Ao acionar esse modo no controle, o código mostrado na Imagem 5 é executado. Nele o Arduino faz a leitura dos dois sensores para o eixo lateral, e dos outros dois sensores para o eixo frontal, faz a subtração de um lado somado ao outro para determinar qual dos lados está mais

próximo a um objeto. Esse valor obtido é somado a 1500, que corresponde ao valor que o rádio transmissor enviaria ao *drone* quando não há movimento.

Imagem 5 – Código responsável pelo modo de voo sensorial

```
if((aux1>1000) && (aux1<1500)){ //modo sensorial
  Input1=s1.ping_cm(); //le sensores
  Input2=s2.ping_cm();
  Input3=s3.ping_cm();
  Input4=s4.ping_cm();

  leftPID.Compute(); // calcula ajustes de PID definidos sobre os valores lidos
  rightPID.Compute();
  frontPID.Compute();
  backPID.Compute();

  ale=1500+Output1-Output2; // faz o calculo para qual lado deve ir
  ele=1500+Output4-Output3;

  aleo.writeMicroseconds(ale); //manda o pulso para o arduino crius
  eleo.writeMicroseconds(ele);
}
```

Fonte: os autores.

Já o modo de voo autônomo se baseia nos dados enviados pelo Raspberry, conciliando-os com os dados obtidos dos sensores a fim de evitar colisões.

O Raspberry Pi envia o sinal determinando aonde o *drone* deve ir: direita, esquerda, acima, abaixo, baseado num sistema de coordenadas bidimensional (*x* e *y*). O campo de profundidade (*z*) é feito por meio do sensor de proximidade frontal; se o objeto encontrado estiver a mais de um metro de distância, o *drone* irá se aproximar dele até ficar a um metro. Caso o objeto se aproxime, o *drone* irá recuar. A Imagem 6 mostra a parte do código responsável por receber os dados enviados pelo Raspberry Pi.

Imagem 6 – Código responsável pelo modo de voo autônomo

```
if (aux1>=1500){ //modo autônomo
  if (InputX > 0)           // Em uma imagem de 320x240: 160 é o meio do eixo x entao quando estiver lendo 160 retornará 0
    OutputX = InputX - 160; // quando estiver lendo 0 retorna -160 e quando ler 320 retornar 160
  if (InputY > 0)           // 120 é o meio no eixo y entao quando estiver lendo 160 retornará 0,
    OutputY = InputY - 120; // quando estiver lendo 0 retorna -120 e quando ler 240 retornar 120

  InputZ=s3.ping_cm(); //verifica a distância do objeto frontal

  if ((InputZ > 0) && (InputZ <= 100)) // se o objeto estiver a menos de um metro o drone fica parado
    InputZ = 0;

  z_PID.Compute();  // faz o ajuste do outputZ

  if (OutputX > 0)
    OutputX = map(OutputX, -160, 160, -300, 300);  // faz os ajustes do outputX
  if (OutputY > 0)
    OutputY = map(OutputY, 240, 0, 100, 800);  // faz os ajustes do outputY

  verificaColisao; // verifica se tem algum objeto no caminho alterando os valores para evitar colidir

  ale=1500+OutputX; // faz o calculo para qual lado deve ir
  thr=1500+OutputY;
  ele=1500+OutputZ;

  thro.writeMicroseconds(thro_in); //manda o pulso para o arduino crius
  aleo.writeMicroseconds(ale);
  eleo.writeMicroseconds(eleo_in);
}
```

Fonte: os autores.

3.4 CONFIGURAÇÃO MULTIWII

O software que irá executar os comandos enviados pelo Arduino Uno é o sistema *open-source* MultiWii, que foi modificado para atender às necessidades do projeto e que roda no Arduino Crius.

Foram feitas diversas alterações no código-fonte do MultiWii, como no tipo de *drone* que está sendo configurado, os canais de recepção dos comandos, a velocidade mínima de rotação dos motores e atribuições dos modos de voo para os comandos auxiliares do rádio transmissor.

3.5 COMUNICAÇÃO ENTRE ARDUINO UNO E RASPBERRY PI

A comunicação entre o Raspberry Pi e o Arduino Uno foi realizada por meio da conexão I^2C. O empecilho de realizar essa conexão é o fato de o Arduino Uno trabalhar com 5 volts nas portas de conexão, enquanto o Raspberry Pi opera em 3,3 volts.

Para contornar essa situação, foi utilizada uma pequena placa que realiza a conversão entre tensões, possibilitando, assim, que os dois dispositivos eletrônicos fossem interligados.

3.6 INTEGRAÇÃO ENTRE ARDUINO UNO E ARDUINO CRIUS

A integração entre os Arduinos foi realizada utilizando-se quatro portas GPIO em ambos os dispositivos.

Cada uma dessas portas fica responsável por uma função: acelerar, girar, eixo lateral, eixo frontal, em que o Arduino Uno envia o valor por essas conexões, e o Arduino Crius faz a leitura, transmitindo os comandos para os motores.

3.7 DRONE MONTADO E PRONTO PARA TESTES

Após realizadas as conexões previamente citadas, foram montados os motores, os controladores de velocidade e instalada a bateria; para finalizar a montagem dos componentes, o *drone* recebeu uma "capa" para proteger os Arduinos, o Raspberry e a câmera. Foram acopladas espumas para absorver os impactos no chão durante os testes.

A Imagem 7 mostra o *drone* pronto após terminado o processo de construção.

Imagem 7 – Construção do *drone* finalizada

Fonte: os autores.

3.8 TESTES

Após a conclusão do desenvolvimento do projeto, iniciaram-se os testes para verificar se o resultado seria conforme o esperado.

3.8.1 **Simulação em software**

Antes de realizar os testes com o *drone* voando, foi utilizado o *software* WinGui para simular o comportamento dos sensores e do algoritmo de visão computacional.

A proposta da simulação foi averiguar se os motores iriam acelerar na direção correta para seguir o objeto, assim como verificar se, ao encontrar um objeto próximo, utilizando os sensores de proximidade, o *drone* iria se mover na direção contrária dele para não colidir.

Na simulação, foi observada a aceleração dos motores do lado esquerdo quando o objeto está no lado direito, fazendo com que o VANT siga o objeto para a direita; e a aceleração dos motores do lado direito quando o objeto está no lado esquerdo, fazendo com que o *drone* se mova para a esquerda.

3.8.2 **Teste de Voo Autônomo**

O teste de voo autônomo, que é o objetivo principal do projeto, foi feito utilizando-se um cilindro de cor amarela o qual o *drone* deveria seguir.

O algoritmo de visão computacional previamente testado em simulação conseguiu identificar a esfera e passar a sua localização para o Arduino Uno, o qual interpretou a localização transformando em instrução de direção, passando-a para o Arduino Crius, que, por sua vez, moveu o *drone* na direção do objeto.

4 **CONCLUSÃO**

O algoritmo de controle de modos de voo foi inicialmente previsto para ser feito no próprio Arduino que comanda e estabiliza o *drone*. Porém o excesso de tarefas o sobrecarregou e, com isso, fez com que não fosse possível ler os sensores, alternar modos de controle e, ainda, controlar o *drone*. A solução foi utilizar um segundo Arduino para desempenhar a tarefa. Com os dois algoritmos desenvolvidos, alterações no software *open source* MultiWii e a construção física do *drone*, ele conseguiu atingir seu objetivo principal, que era seguir um objeto colorido em um ambiente controlado.

Após a conclusão deste trabalho, fica claro que a visão computacional pode ser utilizada para automatizar não somente veículos aéreos não tripulados, mas diversas áreas em que é necessário "enxergar" o mundo e transformá-lo em algoritmos e programas.

Development of an autonomous drone guided through computer vision techniques

Abstract

This paper presents the development process of a drone with ability to move autonomously, using, for that, computer vision techniques. The project is based on the Arduino platform, and the Raspberry Pi microcomputer, using the Linux operating system, as well as the C++ programming language and the open source computer vision library OpenCV. As well as computer vision (set of techniques which intend to provide machines the capacity of "seeing"), the drones, initially developed with military proposals, are increasingly gaining space in several other areas. With this, an intelligent system is considered necessary in order to be operated autonomously, using them more efficiently without the need for human control, speeding the areas where they are used. In the autonomous system, the drone flies and performs operations by processing the videos captured by a camera, passing instructions and directions for the computer, which directs the drone to the desired location. At the end of the project, it will be presented an autonomous drone capable of executing instructions by

recognizing a colored object in a controlled environment.

Keywords: Computer vision. Drone. Arduino. Raspberry Pi. OpenCV.

REFERÊNCIAS

BANZI, Massimo. **Getting Started with Arduino**. Sebastopol, California: O' Reilly Media Inc, 2008.

BRADSKI, Gary; KAEHLER, Adrian. **Learning OpenCV. Computer Vision with OpenCV Library**. Sebastopol, California: O' Reilly Media Inc, 2008.

GARRET, Filipe. **O que é drone e para que serve? Tecnologia invade o espaço aéreo**. 2013. Disponível em: <http://www.techtudo.com.br/noticias/ noticia/2013/10/o-que-sao-e-para-que-servem-os-drones-tecnologia-invade--o-espaco-aereo.html>. Acesso em: 20 ago. 2014.

HAMANN, Renan. **Drones**: o que é preciso saber sobre os robôs voadores [ilustração]. 2013. Disponível em: <http://www.tecmundo.com.br/ veiculos/41834-drones-o-que-e-preciso-saber-sobre-os-robos-voadores-ilustracao-.htm>. Acesso em: 20 ago. 2014.

KUGELSTADT, Thomas. **Digital Interface – Understanding the I2C bus**. Estados Unidos: Texas Instruments Inc, 2009.

MCROBERTS, Michael. **Getting Started with Arduino**. Nova York: Apress, 2010.

MULTIWII. **MultiWii**. 2014. Disponível em: <http://www.multiwii.com/wiki/index.php?title=MultiWii>. Acesso em: 20 ago. 2014.

RICHARDSON, Matt; WALLACE, Shawn. **Getting Started with Raspberry Pi**. Sebastopol, California: O' Reilly Media Inc, 2013.

SCHMIDT, Maik. **Raspberry Pi a Quick-Start Guide**. Estados Unidos: Pragmatic Expression, 2012.

STROUSTRUP, Bjarne. **The C++ Programming Language – Fourth Edition**. Michigan, 2013.

TAUFER, Patrícia. **Em São Paulo, drone** são usados para ajudar a vender apartamentos. 2013. Disponível em: <http://g1.globo.com/bom-dia-brasil/noticia/2013/09/em-sao-paulo-drones-sao-usados-para-ajudar-vender-de-apartamentos.html>. Acesso em: 20 ago. 2014.