Robô Seguidor de Linha com Recurso a uma RaspiCam

1st André Moreira Oliveira
Dept. de Engenharia Eletrotécnica
ISEP - Instituto Superior de Engenharia do Porto
Porto, Portugal
1181045@isep.ipp.pt

2nd Tiago Cunha
Dept. de Engenharia Eletrotécnica
ISEP - Instituto Superior de Engenharia do Porto
Porto, Portugal
1180922@isep.ipp.pt

***Abstract*—O projeto em causa procura desenvolver uma implementação de um sistema dinâmico e devidamente autónomo que promova o movimento de um veículo com rodas diferenciais segundo processos computacionais alocados a vários nós ROS (*Robot Operating System*). A Raspberry Pi 3 a bordo do veículo vem equipada com uma raspicam capaz de extrair as diversas informações do percurso. O processamento destas imagens deve resultar numa atuação precisa nos motores acoplados a cada roda. A comunicação com Raspberry Pi é conseguida através de *Secure Shell* (SSH) *protocol* permitindo uma ligação remota na mesma rede.**

***Index Terms*—ROS, Raspberry Pi, Raspicam, WiringPi, raspicam_node, tele_op_keyboard, geometry_msgs**

I. Introdução

O conhecimento preciso da localização de um sistema robótico móvel é essencial para a realização de várias tarefas automatizadas. O método de navegação utilizado no sistema robótico, para a aquisição em tempo real, da posição deste vai além deste projeto. Contudo, para uma condução autónoma típica de um *line following robot* é necessário a conjugação de diversos fatores. O conhecimento da sua orientação, velocidade linear e angular permitem o movimento controlado fruto de um processamento apropriado.

A. Objetivos

Em causa está um projeto possibilita uma oportunidade de explorar e desenvolver *software* na *framework Robot Operating System* (ROS) segundo uma implementação física de um veículo com rodas diferencias controlado por uma Raspberry Pi 3b+ além da familiarização com a mesma. Como tal, procura-se atingir os seguintes pontos:

- Desenvolvimento de um *ROS node* (tele_op_keyboard) para controlo da plataforma por teleoperação tendo por base o *package* teleop_twist_keyboard a fim de publicar num tópico mensagens do tipo geometry_msg/Twist;
- Desenvolvimento de um *ROS node* (motor_control) para controlo dos motores por *Pulse Width Modulation* (PWM). Para tal deve ser subscrito o tópico publicado pelo *node* que permite o controlo da plataforma por teleoperação;
- Desenvolvimento de um *ROS node* (image_processing) para controlo da plataforma por seguimento de uma linha preta num cenário branco, segundo uma raspicam. Este *node* deve publicar no mesmo tópico do *node* tele_op_keyboard mensagens do tipo geometry_msg/Twist, devendo no final fazer a sua total substituição;
- Controlo dos motores através do mecanismo *proportional–integral–derivative controller* (PID) para uma resposta mais suave face ao erro detetado;
- Seleção minuciosa dos diversos parâmetros associados ao algoritmo de processamento de imagem;
- Adaptação do *node* image_processing para publicação num tópico a imagem processa para fins de processos de *debugging*;

B. Plataforma

Obstante aos detalhes técnicos da seleção dos elementos que constituem este robô é possível enumerar os principais componentes que compõe a plataforma presente na Figura 1 e como tal permitem o desenvolvimento e implementação de algoritmos de navegação. Assim, o robô conta com [1]:

- Raspberry Pi 3b+ localizada na frente do robô parte superior;
- Raspberry Pi *Camera Module v2.0* e o respetivo suporte ligeiramente à frente do *onboard computer*;
- Raspberry Pi *Motor Driver Board* devidamente acoplada entre a *onboard computer* e os *DC motors* responsável pelo controlo destes últimos;
- Dois *Direct Current* (DC) *motors* e respetivas rodas;
- Dois encoders não implementados a nível de *software*;
- Bateria Zippy Compact 25C Series Li-PO 2700 mAh, para alimentação do sistema, posicionada na traseira do robô;
- Uma *castor wheel* centrada aos *DC motors* mas na parte posteriores do robô;
- Interruptor on/off para o devido controlo do sistema;
- Suporte para toda a plataforma;

Relativamente aos componentes de *software*, a Raspberry Pi possui uma distribuição Linux Ubuntu 18.04 (Bionic) LTS4 fruto de uma imagem Ubiquity Robotics Raspberry Pi localizada num catão SD. A versão em causa desenvolvida pela Ubiquity Robotics [2] conta com ROS Melodic Morenia instalado [3].

Por sua vez, toda a interação com os *General Purpose Input/Output* (GPIO) é conseguida através da biblioteca WiringPi [4], responsável pela camada de abstração ao utilizador.

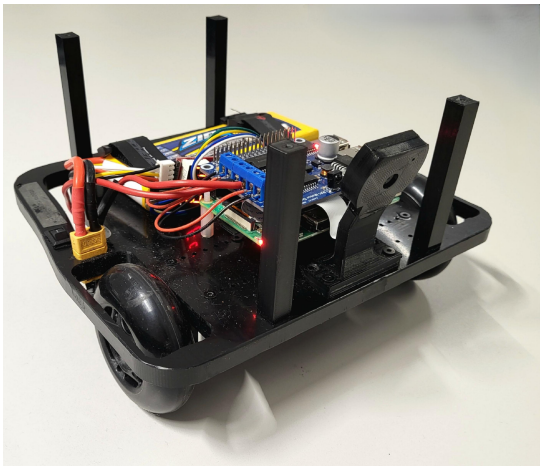


Fig. 1: Veículo com rodas diferencias utilizado, Raspberry Pi, *Motor Driver Board* e Raspicam [1]

Uma arquitetura generalizada do *hardware* do robô pode ser encontrada na Figura 2. Nesta é possível verificar a interligação dos diferentes intervenientes e como tal assegurar a compreensão dos conceitos que serão abordados ao longo do artigo.

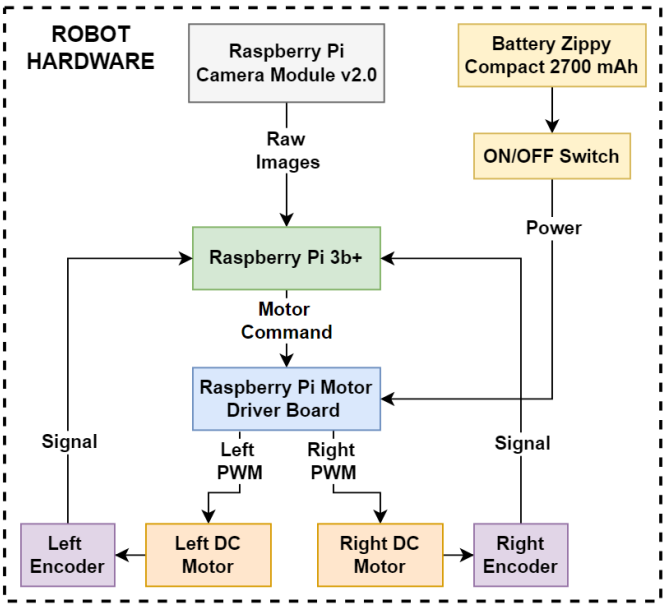


Fig. 2: Arquitetura generalizada do *hardware* da plataforma robótica

C. Estrutura do Artigo

O presente artigo encontra-se dividido em quatro capítulos capazes de descreverem todo o estudo realizado até então.

A organização deste artigo segue uma lógica evolutiva do projeto final. Como tal, no presente capítulo é feita uma abordagem genérica ao projeto assim como realçadas as partes integrantes do desenvolvimento e da plataforma.

O capítulo seguinte reúne a implementação propriamente dita, na medida em que é apresentado o diagrama dos pacotes presentes no funcionamento normal do ROS, a lógica adotada para o controlo dos motores e todos os recursos e tecnologias exploradas.

Reúne-se no capítulo posterior as experiências e resultados verificados como consequência de todo o processamento de imagem realizado.

Finaliza o presente artigo com um capítulo de conclusões com uma síntese da implementação conseguida e potenciais avaliações finais.

II. Implementação do Projeto

O projeto em causa levou a um estudo de diferentes tecnologias que permitissem a interação com o *hardware* disponibilizado. Tem por base a plataforma ROS numa vertente dinâmica que permita o aprofundamento de vários conhecimentos. Seja pela programação dos GPIO da Raspberry através da biblioteca WiringPi e uso de outros *packages*, *i.e.* tele_op_keyboard e raspicam_node, mas também pelas tecnologias que permitem toda a interação na *ROS Network* criada.

A. Diagrama dos Pacotes

O ROS é um conjunto de bibliotecas de software e ferramentas para o desenvolvimento em aplicações robóticas. Agrupa diversas vantagens, permitindo a reutilização de código base, facilidade de incorporação de código *open-source* fruto da sua adaptabilidade a nós escritos em Python e C++, além de boas ferramentas de simulação, *e.g.*, Rviz e Gazebo, para um custo computacional baixo [5].

O ROS assume três níveis de conceitos, porém, para o projeto em causa apenas serão abordados os principais recursos inerentemente utilizados. Contudo, para maiores detalhes é possível verificar a página oficial [6].

1) *ROS Filesystem Level*
   - *Packages*: principal ferramenta do ROS para gestão do *software*, podendo conter, entre outras coisas, *ROS runtime processes (nodes)*, *ROS-dependent library*, *datasets* e ficheiros de configuração;
2) *ROS Computation Graph Level*
   - *Nodes*: processos que realizam algum tipo de computação, podendo comunicar com outros *nodes*, sendo escritos com uso de uma *ROS client library*, como roscpp ou rospy.
   - *ROS Master*: atua como um *name registration*, permitindo aos *nodes* neste presente se encontrarem, trocarem mensagens e invocar serviços;
   - *Messages*: estrutura de dados utilizada na troca de dados entre nodes, podendo conter *primitive types variables*, *e.g.*, *integer, floating point, boolean*, etc.

- *Topics*: As mensagens são controladas através de um modelo *publish / subscribe*. Um *node* envia uma mensagem publicando-a num determinado tópico. Por sua vez, um *node* pode receber mensagens subscrevendo um determinado tópico.

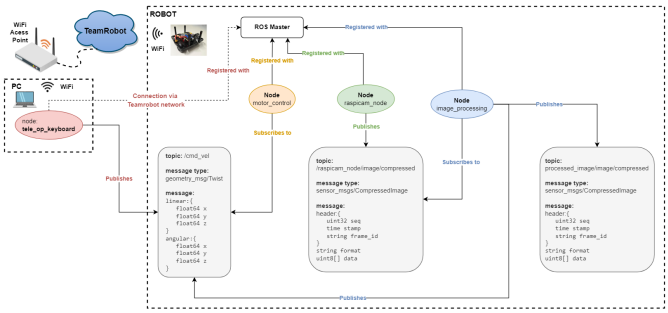


Fig. 3: *Package diagram: nodes, topics* e *menssages*

O diagrama de pacotes presente na Figura 3 pode ser visto como um *unidirectional pipeline*. Numa primeira fase e para efeitos de testes de todo o *hardware* apostou-se no desenvolvimento de um *node* tele_op_keyboard com permissões para execução localmente no computador e publicação no tópico /cmd_vel. Este permitia o controlo do robô através dos *inputs* provenientes do teclado do computador que o executasse. A Tabela I sintetiza o efeito da seleção de cada uma das teclas apresentadas. Como tal espera-se a atuação correspondente.

TABLE I: Correlação entre o teclado e o movimento do robô

Keyboard	Linear Velocity	Angular Velocity	Motion
u	0.5	1	↖
i	0.5	0	↑
o	0.5	-1	↗
j	0	1	↺
k	0	0	STOP
l	0	-1	↻
m	-0.5	-1	↙
,	-0.5	0	↓
.	-0.5	1	↘

O sistema foi configurado para operar com apenas um ROS Master, do lado da Raspberry Pi mas de forma a permitir a operação com vários dispositivos. Este facto permite que os *nodes* presentes no computador interajam com os *nodes* em execução na *onboard computer* desde que esteja acessíveis na mesma rede. Este facto permitiu a *teleoperation*, visualização do *robot status* e análise do processamento da imagem em tempo real através do tópico processed_image/image/compressed criado.

Do lado da Raspberry Pi podemos ver três *nodes* em execução, nomeadamente o motor_control para atuação dos motores faces às mensagens de dados extraídas do tópico /cmd_vel subscrito. Por sua vez, os dados obtidos do sensor de imagem são publicados através do *node* raspicam_node no tópico /raspicam_node/image/compressed. Os mesmos sofrem um processamento de imagem focado na deteção dos limites linha preta e estabelecimento de uma potencial direção de circulação do robô (auxiliada de uma correção pontual recorrendo a um PID) através da substituição completa do node tele_op_keyboard com a publicação no tópico /cmd_vel.

O algoritmo de processamento de imagem e controlo dos motores será discutido nos próximos capítulos deste artigo.

B. Controlo dos Motores

O robô explorado encontra-se equipado com dois *DC motors*. Estes dispositivos eletromagnéticos são capazes de converter energia elétrica em energia mecânica, nomeadamente a rotação do seu veio. Este é composto por enrolamentos percorridos por linhas de fluxo do campo magnético que devem pontualmente repelir e atrair os ímans estáticos no seu interior. Esta força gerada depende fortemente da corrente a este fornecida. A velocidade de rotação de é proporcional à tensão aos terminais dos enrolamentos. Como tal, recorrendo a uma onda PWM devidamente gerada por um *motor driver* é possível controlar a sua velocidade e sentido de rotação [7].

Esta onda é aplicada sincronizadamente aos terminais dos motores M1, M2, M3 e M4, sendo os primeiros referentes ao motor esquerdo e os restantes ao motor direito. A tabela II reúne os diferentes estados de cada um dos pinos para a devida atuação dos motores. Através da biblioteca WiringPi é possível desenvolver um *ROS node* (motor_control) e controlar os parâmetros anteriormente referidos fruto de um PWM variável conforme as necessidades.

A biblioteca WiringPi simplifica a programação e configuração dos periféricos da Raspberry, nomeadamente com as funcionalidades que permitem gerar os pretendidos sinais PWM [8].

TABLE II: Correspondência entre os pinos e o movimento dos motores [1]

M1	M2	M3	M4	Description
0	1	0	1	Motors rotate forward, robot goes straight
1	0	1	0	Motors rotate backwards, robot draws back
0	0	0	1	Right motor rotates forward, robot turns left
0	1	0	0	Left motor rotates forward, robot turns right
0	0	0	0	Motors stop, robot stops

O PWM gerado para cada motor segue uma relação trigonométrica face ao angulo publicado no tópico /cmd_vel. É assumido uma velocidade base para os cálculos do PWM variável. Contudo, o ângulo recebido encontra-se limitado $\pm$ 45 $^\circ$ por efeitos de segurança. Nos movimentos frente, trás é aplicada a velocidade linear desejada, nos movimentos direita e esquerda é aplicada a velocidade angular desejada ao motor contrário ao sentido que se pretende efetuar a rotação. Nos restantes movimentos diagonais, *e.g.*, frente-esquerda, trás-direita, etc., a velocidade entre o motor direito e esquerdo encontra-se condicionada por uma relação trigonométrica que atua nos 90 $^\circ$ do primeiro quadrante. Um exemplo prático, numa trajetória que exija um movimento para a frente mas com um deslocamento ligeiro para a direita (frente-direita) é necessário atuar com maior velocidade o motor esquerdo, logo o PWM neste motor seria diretamente proporcional à função cosseno deslocada de 45 $^\circ$ no seu argumento, dado que o

angulo recebido seria negativo. Por sua vez, o PWM no motor direito seria diretamente proporcional à função seno deslocada de 45 °.

C. Tratamento e Processamento de Imagem

Conforme mencionado, a raspicam a bordo vai ser responsável por extrair as diversas informações do percurso a percorrer. Portanto um preciso tratamento e processamento de imagem é essencial para o correto funcionamento do *line following robot*.

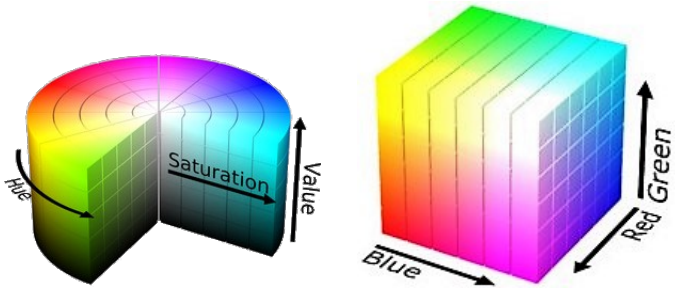
O *package* raspicam_node possui diversos *ROS nodes* que devem criar vários tópicos representativos da informação extraída da Raspberry Pi Camera Module v2.0. Conforme descrito anteriormente o tópico /raspicam/image/compressed possui a imagem comprimida publicada por um destes *nodes*. Ao executar o *launch file* `roslaunch raspicam_node camerav2_410x308_10fps.launch` é possível obter os vários *frames* capturados pela câmara para futuro tratamento. A 10 *frames per second* (fps) obtemos dez imagens por segundo para uma resolução consideravelmente menor que a resolução máxima da câmara.

Posteriormente ao processamento é possível visualizar no rqt_image_view a transmissão da imagem original e a imagem trabalhada em tempo real.

Naturalmente, antes de processar as imagens é necessário efetuar tratamento de modo a filtrar tudo o que não é a linha preta a seguir. A *pipeline* deste procedimento encontra-se representada na Figura 5. A primeira etapa consiste no acesso à imagem comprimida publicada no tópico /raspicam/image/compressed e descomprimi-la utilizando a função `imdecode()` da biblioteca OpenCV que retorna a imagem devidamente descomprimida (primeira imagem da Figura 5).

Seguidamente é selecionado uma *region of interest* (ROI). Dado que o robô quando se movimenta em linha reta, a sua velocidade aumenta, esta região vai se adaptar consoante a velocidade do robô. No exemplo da Figura 5, como o robô estava a movimentar-se a uma velocidade alta, a região de interesse selecionada da primeira imagem é a zona B, demarcada a azul, no entanto para os casos em que o robô se movimenta a uma velocidade mais baixa é selecionada a zona A, demarcada a vermelho.

Os processos seguidamente descritos assumem o modelo *Hue Saturation Value* (HSV) na representação das cores da imagem. Análogo ao *Red Green Blue* (RGB), é possível representar um espaço de cores. A Figura 4 apresenta uma distinção gráfica da forma de segmentação das três componentes que representam cada um dos modelos. A variação da saturação permite introduzir sombras de cinzentos ou quando completamente saturado omitir a componente branca. Enquanto o parametro *value* permite variar o brilho ou a intensidade da cor. Tipicamente é mais completo segmentar objetos, neste caso distinguir a cor preta das restantes presentes no cenário no *RGB colorspace*. Contudo, a biblioteca OpenCV lê as imagens em formato BGR (*Blue Green Red*), uma opção histórica assumida no momento de desenvolvimento da biblioteca fruto da compatibilidade entre os fornecedores de câmaras e o desenvolvedores de *software* [9].



(a) *HSV model* (b) *RGB color model*

Fig. 4: Modelos gráficos para representação dos três canais dos modelos HSV e RGB [10]

Após a escolha da região de interesse, é executado uma série de transformações a partir da biblioteca OpenCV na imagem de modo a filtrar tudo o que não é preto da imagem. A primeira transformação é dada pela função `cvtColor()` [11]. Esta função converte uma imagem BGR para HSV (terceira imagem da Figura 5). Seguidamente é aplicado uma técnica de filtragem não linear à imagem anterior de modo a suavizar imperfeições (quarta imagem da Figura 5), a partir da função `medianBlur()` responsável pela substituição um dado pixel pela mediana de todos os pixeis contidos na área do kernel destacada [12]. Posteriormente, com recurso à função `inRange()` [13], a quarta imagem passa por um processo de binarização, dada uma *mask* especifica para separação do preto das restantes cores. Em seguida é processado apenas os contornos da imagem a partir da função `adaptiveThreshold()` [14] (sexta imagem da Figura 5). Finalmente, para efeitos de *debugging* é desenhado na imagem os pontos intermédios detetados, a linha de direção e o ângulo de saída do PID no canto superior esquerdo (sétima imagem da Figura 5).

Seguido do tratamento de imagem será necessário relacionar os dados extraídos com o movimento a efetuar. A Figura 6 esquematiza todo processo posterior à imagem tratada, desde a identificação dos pontos associados à mudança de cor preta para branco até à publicação das velocidades linear e angular devidamente corrigidas.

A imagem resultante do tratamento é uma matriz de valores, sendo esses correspondentes à cor do pixel naquela posição. Tendo em conta este facto e de que, por conta do processo de binarização, a imagem apenas possui duas cores. É então, facilmente, detetado o início e fim da linha a seguir. Em específico, este processo consiste na análise da matriz de valores da imagem de modo anotar pontos onde se dá a transição da cor preta para a cor branca.

Seguidamente, para garantir que os pontos anotados correspondem aos contornos da linha, é verificado se a distância entre os pares de pontos identificados se encontra numa margem típica da linha a seguir.

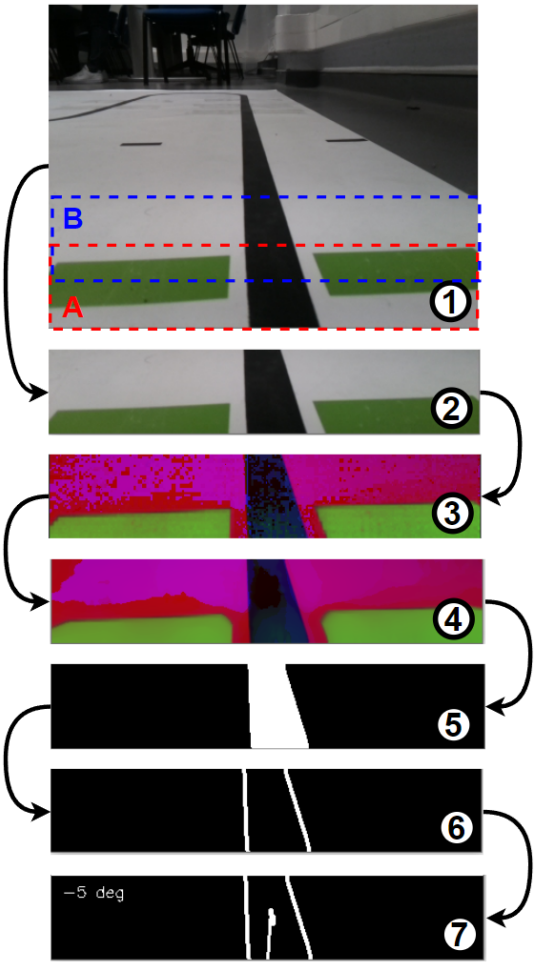


Fig. 5: *Pipeline* do tratamento de imagem

Posteriormente, a partir dos pontos anteriormente encontrados, é calculada a média para cada par de pontos numa linha. Dos pontos intermédios calculados é utilizado o ponto mais distante para calcular o ângulo α da posição deste relativamente à posição zero central. O valor deste ângulo, pela atuação de um controlador PID devidamente dimensionado, é corrigido de modo proporcionar um controlo do robô mais suave e responsivo nos vários contornos do trajeto.

Finalmente, o a velocidade angular anteriormente estipulada pelo PID é publicada no tópico /cmd_vel, nomeadamente como `angular.z` da mensagem Twist. Adicionalmente, quando o erro atual e o erro anterior estiver a baixo de um dado *threshold* típico de uma reta, o robô acelera para uma velocidade superior e assume a região de interesse B (região representada na primeira imagem da Figura 5).

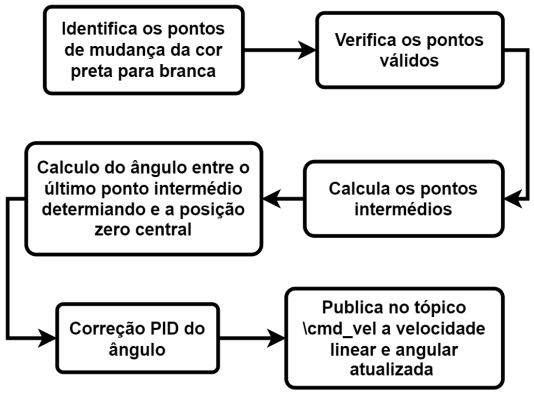


Fig. 6: *Pipeline* do processamento de imagem.

III. Experiências e Resultados

O percurso de estudo encontra-se presente na Figura 7. Nem todos os testes foram realizados na pista apresentada. Contudo, é importante realçar que a tonalidade preta da linha a seguir facilitou consideravelmente a deteção desta face à aplicação de uma simples fita isoladora preta. O local de teste encontrava-se sujeito a bastante luz natural e quando esta se reduzia, a luz artificial atua. Como tal, era necessário uma calibração da *mask* que realiza o `adaptiveThreshold()` no momento do teste, caso contrário podíamos ver a deteção das linhas severamente condicionada. Por sua vez, os elementos de diferentes cores em torno da pista não apresentaram qualquer tipo de problema no processamento atual. O mesmo podia ver-se condicionado para um processamento somente formato BGR ou Gray-Scale, exigindo a substituição dos elementos anômalos por cores brancas.

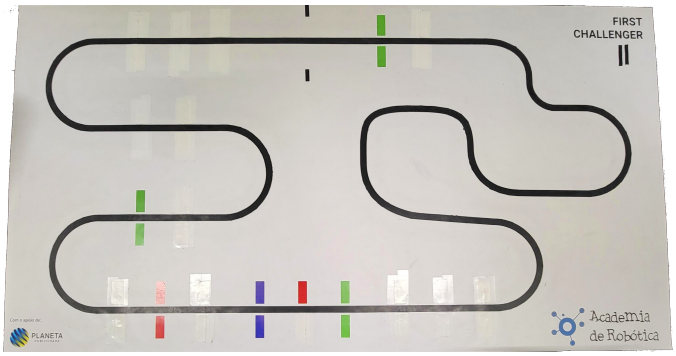


Fig. 7: Circuito para teste do *line following robot* [2]

O cálculo do angulo de atuação é obtido através da função atan2 uam vez que tem em consideração as possibilidades dadas pelos quatro quadrantes do círculo trigonométrico. Assim sendo, α é dado pela seguinte equação:

$$\alpha = \text{atan2}(y_2 - y_1, x_2 - x_1) \tag{1}$$

Uma demonstração prática está presenta na Figura 8 onde é possível ver a representação do ângulo α para a posição zero central (x_1, y_1) e o ponto mais distante (x_2, y_2).

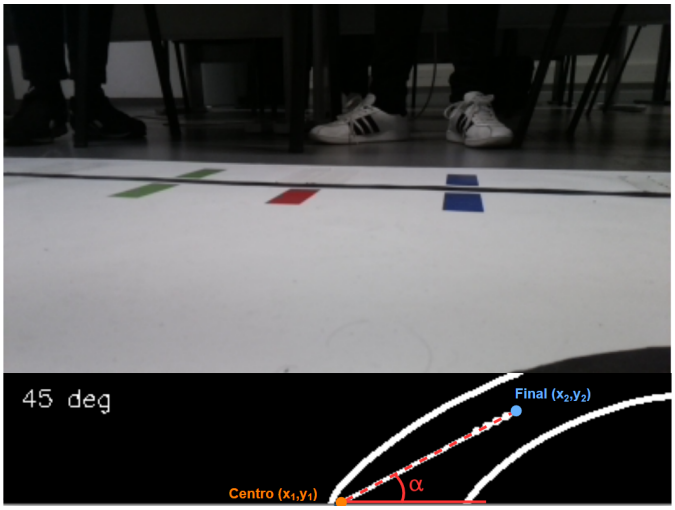


Fig. 8: Análise matemática do processamento de imagem

O PWM variável segundo relações trigonométricas viu-se eficaz. Para os motores em causa verificamos uma aceleração linear máxima de aproximadamente 0,5 m/s contudo, este valor pode variar consoante a tensão da bateria a bordo do veículo e das irregularidades do pavimento que alteração o fator tração durante o movimento.

No repositório GitHub presente no capítulo das conclusões pode ser encontrado dois vídeos do robô a percorrer a pista apresentada nos diferentes sentidos juntamente do processamento de imagem observado em tempo real.

IV. Conclusões

Um *line following robot* deste tipo recorrendo a um sensor de imagem apresenta diversas possibilidades de implementação. Na solução desenvolvida foram criados três *nodes* e reutilizado o *package* da raspicam_node dos quais resulta em três *publishers* e dois *subscribers*.

O tratamento de imagem exige uma calibração da *mask* que realiza o `adaptiveThreshold()` face às condições de iluminação no momento do teste. Contudo, as linhas de diferentes cores não foram problema o processamento atual.

O cálculo do angulo corrigido fruto da aplicação do controlador PID pode surgir condicionado para valores diferentes de bateria no robô, como tal foram deixadas em comentário alguns *sets* que podem servir de base de calibração para diferentes velocidades.

A definição de um ROI variável permitiu a otimização da circulação nas retas, análogo ao verificado nas plataformas convencionais dos sistemas de navegação automóvel que resulta num *zoom out* do mapa.

A biblioteca OpenCV mostrou-se trivial neste projeto na medida em que permitiu condicionar facilmente a extração de dados de cada imagem, num curto intervalo de tempo, garantido a resposta praticamente em tempo real.

Por fim, e finalizando este projeto, os principais ficheiros associados à criação, teste e desenvolvimento de todo o *software* estão disponíveis, em formato *open-source* no seguinte repositório do GitHub: https://github.com/AndreOliveira00/Line-Following-Robot.git.

References

[1] Academic, “Robot Platform Manual,” pp. 3–44, 2021.
[2] “Ubiquity Robotics - Get To Your Robot Application 2 Years Faster.” https://www.ubiquityrobotics.com/ (accessed May 04, 2022).
[3] ROS.org, “Melodic - ROS Wiki,” 2018. http://wiki.ros.org/melodic (accessed May 04, 2022).
[4] “WiringPi.” http://wiringpi.com/ (accessed April 28, 2022).
[5] ”Why ROS? - ROS.org” https://www.ros.org/blog/why-ros/ (accessed April 07, 2022).
[6] “ROS/Concepts - ROS Wiki.” http://wiki.ros.org/ROS/Concepts (accessed May 08, 2022).
[7] A. Elaborados, P. O. R. Betina, and C. Neves, “Melec máquinas de corrente contínua.”
[8] “WiringPi.” http://wiringpi.com/ (accessed: April 08, 2022).
[9] “Why OpenCV Using BGR Colour Space Instead of RGB - Stack Overflow.” https://stackoverflow.com/questions/14556545/why-opencv-using-bgr-colour-space-instead-of-rgb (accessed May 07, 2022).
[10] OpenCV community, “OpenCV: Thresholding Operations using inRange,” 2018. https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html (accessed May 07, 2022).
[11] ”Python — cv2 cvtColor() Method” - Java2Blog. Retrieved April 28, 2022, from https://java2blog.com/cv2-cvtcolor-python/
[12] ”OpenCV - Median Blur”. Retrieved April 28, 2022, from https://www.tutorialspoint.com/opencv/opencv_median_blur.htm
[13] ”OpenCV: Thresholding Operations using inRange”. Retrieved April 28, 2022, from https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html
[14] ”OpenCV - Adaptive Threshold”. Retrieved April 28, 2022, from https://www.tutorialspoint.com/opencv/opencv_adaptive_threshold.htm