Controlador de Semáforos

Relatório Técnico Final - Laboratório de Aquisição e Controlo

Pedro Sousa (up201704307)

Leonardo Hügens (up201705764)

1 Introdução

Neste relatório estará documentado o processo de criação de um controlador de semáforos. Para isso, utilizamos o LabView como ferramenta principal, também utilizando um pouco de Python na automatização.

Em primeiro lugar, começamos a discutir que situação iríamos estudar. Começamos por pensar na interceção da Rua Júlio Dínis com a praça de Mouzinho de Albuquerque pois apresenta uma interação interessante entre o semáforo dos peões e os semáforos de entrada e saída da rotunda. No entanto, decidimos estudar um cruzamento pois abria possibilidades para novas interações. O cruzamento em questão é o cruzamento entre a Rua Latino Coelho e a Rua Alegria.

2 Especificações iniciais

(...)

3 Planeamento

Para este projeto, implementamos um processo de evolução natural de trabalho. Isto é, começamos com um programa simples (α_0.1) e fomos inserindo novas características e funcionalidades. Apresentamos agora um breve resumo das várias fases do programa.

- α_0.1: Sistema com semáforo principal e semáforo para peões perfeitamente sincroonizados. Apresenta um botão de emergência que, caso esteja ativo, o semáforo para peões é desativado e o semáforo principal fica em amarelo intermitente.
- α_0.2: Reforma à primeira versão do programa. Apresenta as mesmas características do α_0.1, porém, de forma mais compacta, assim, quando fossemos estudar sistemas mais complexos teríamos uma maneira simples de fazer semáforos usando uma subVI de α_0.2.

- α_0.3: Com o insucesso de α_0.2, fazemos a nossa segunda reforma. Processo utiliza uma ideia semelhante à versão 0.2 mas agora fizemos um semáforo à volta de "case structures". Apresenta dois semáforos principais, conectados como se fossem dois semáforos de um cruzamento, e os respetivos semáforos para peões. Apresenta também um mecanismo de acionameto antecipado por controlo de velocidade dos veículos e um mecanismo que, caso haja bastante tráfego na via, o semáforo utiliza essa informação de forma a haver uma desiguldade "positiva"[1] entre o tempo em que está em vermelho e o tempo em que está no verde + amarelo, e assim diminuir naturalmente o tráfego.

- α_0.4: Apenas uma remodelação do α_0.3. Agora apresenta uma representação visual do cruzamento que estamos a estudar de forma a tornar mais direta a sua leitura.

4 Desenvolvimento

Nesta parte do relatório iremos explicar em detalhe as fases já apresentadas do nosso projeto.

α-0.1

Em primeiro lugar, criamos o semáforo principal, para isso utilizamos booleanos do tipo botão, alterando a cor para o respetivo elemento do semáforo e fizemos uma flat sequence onde por cada frame apenas uma luz estava ligada. Depois criamos mais dois botões referentes ao semáforo para peões e alteramos a sua condição consoante o estado em que o semáforo principal estava (se o semáforo estava vermelho o semáforo para peões ficava verde, (se o semáforo estava verde o semáforo para peões ficava vermelho). Por fim criamos uma case structure que, caso o botão de emergência estivesse ativo corria um ciclo while em que a luz amarela ligava e desligava e caso estivesse desligado corria o nosso programa anteriormente escrito. Por fim, introduzimos um timer que apresenta ao peão quanto tempo falta para o seu semáforo ficar vermelho.

[1] Com desigualdade "positiva" queremos dizer que o semáforo vai estar mais tempo no verde+amarelo do que no vermelho

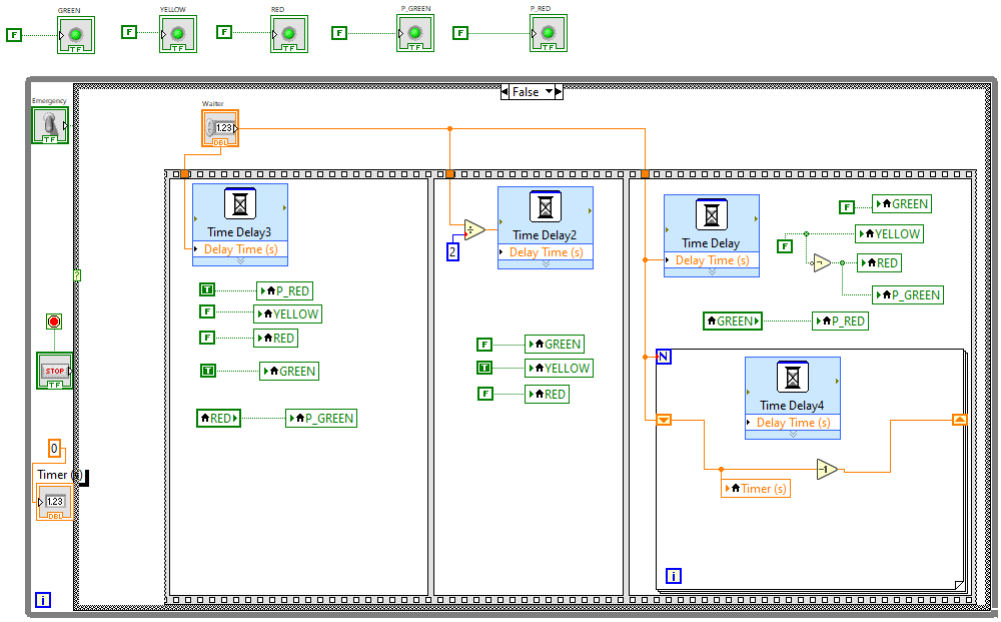

Figura 1: $\alpha\_0.1$

α-0.2

Refiro aqui a imagem 2

Figura 2: Breve descrição.

α-0.3

A ideia inicial era a implementação de um "state" que está associado ao semáforo, isto é:

- Se o semáforo está no estado 0, ele está vermelho e passa para o estado 1 quando se tiver passado no mínimo[2] um tempo $t = timer$.[3]
- Se o semáforo esá no estado 1 ele começa verde e, passado um tempo $t = timer$, muda para amarelo, onde permanece um tempo $t = \frac{timer}{2}$, por fim, muda para o estado 0.

[2] este "no mínimo" é interessante e será discutido mais à frente
[3] timer é um input do nosso sistema de forma a definir o tempo que o semáforo fica no verde

Aplicando isto a um semáforo simples o que acontece é exatamente o esperado, e apenas tivemos de usar um case structure para definir os estados e uma flat sequence para fazer o semáforo ficar verde e depois amarelo.

Agora, decidimos emparelhar outro semáforo. Para isso, estudamos um cruzamento simples entre duas vias de sentido único. Neste exemplo, tudo está em sincronia:

- Quando um semáforo está a permitir passagem (estado 1), o outro não permite (estado 0).
- Quando um semáforo está a permitir passagem (estado 1), o seu semáforo de peões associado não permite.

Para implementar esta ideia fizemos com que os estados dos semáforos estivessem conectados. Isto é, o semáforo 1 está num estado X, então o semáforo 2 está no outro estado.

Mas, com esta teoria deparamo-nos com um problema conceptual. Um semáforo fica no estado 0 um tempo $t = timer$ e fica no estado 1 um tempo $t = 1.5timer$ (1.0 no verde e 0.5 no amarelo). Se os semáforos estão perfeitamente conectados quanto tempo irá o semáforo estar no vermelho? A resposta é $1.5timer$ o que é o esperado, porque o estado só muda quando tudo o que tiver de acontecer naquela iteração do *for* (que é o nosso *for* temporal) terminar, ou seja, $1.5timer$ depois.

Para finalizar, adicionamos duas novas ferramentas:

- Um mecanismo que, caso haja um carro a mais de $50Km/h$, o amarelo é acionado antecipadamente. No entanto, o semáforo fica no verde no mínimo um tempo $t = \frac{timer}{2}$. Este tempo foi introduzido para quando o semáforo fica verde não mudar imediatamente para amarelo, havendo sempre um delay justificado.
- Um mecanismo que, caso haja um enorme tráfego de carros, o semáforo permaneça menos tempo no vermelho de forma a haver um melhor fluído de tráfego.

Estes dois mecanismos estão ligados pois, no fundo, acionar um aviso de congestionamento na via 1 é o equivalente a ligar o mecanismo de velocidade na via 2, pois, se na via 2 está menos tempo no verde, por consequência, na via 1 está menos tempo no vermelho.

Adicionamos uns sliders para representarem o tráfego nas vias e fizemos com que o seu valor diminuísse quando está verde e amarelo e aumentasse quando está vermelho. Adicionamos também um input "rácio de entrada" que, se saem Y carros enquanto está verde, entram "rácio de entrada" $\times$Y carros enquanto está vermelho. Assim, podemos associar este valor, por exemplo, à altura do dia. Se, na hora de ponta entram $5\times$ mais carros do que saem, à noite entram $\frac{1}{10}\times$ dos carros que saem até que não há trânsito de todo.

α-0.4

Nesta implementação decidimos estudar o caso escolhido, ou seja, o cruzamento da rua Latino Coelho e da rua da Alegria. Para isso, remodelamos o $\alpha\_0.3$ de forma a queria uma melhor visualização do cruzamento no Front Panel do LabView.

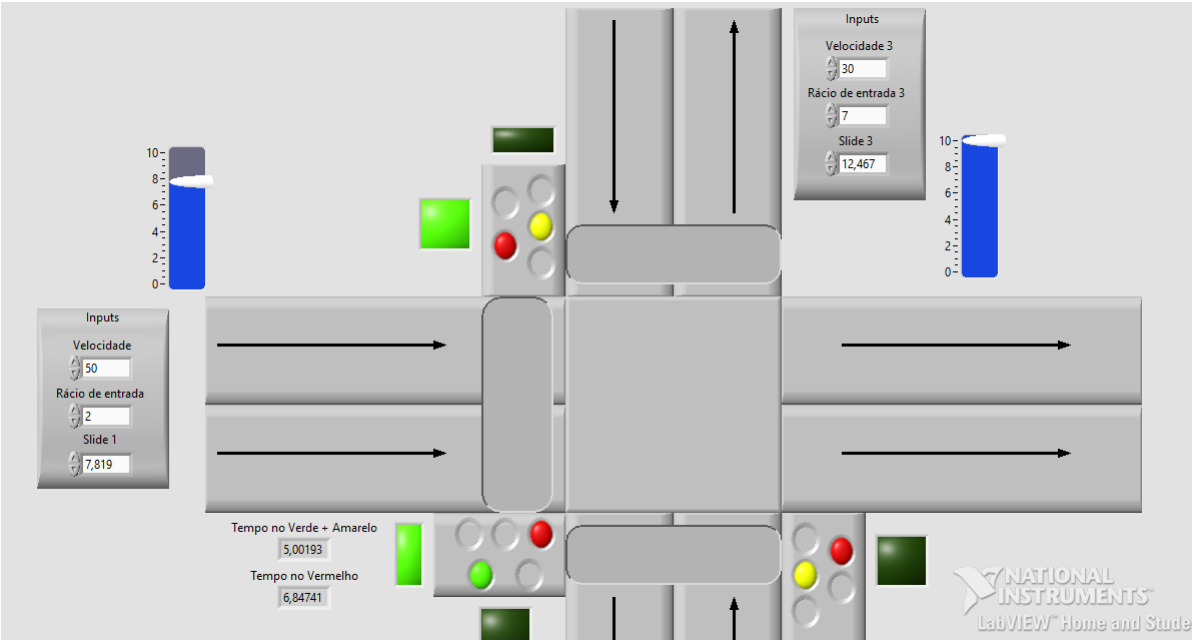

Figura 3: Front Panel do $\alpha\_0.4$ com o esquema do cruzamento entre a rua Latino Coelho e a rua da Alegria; o botão retangular está ativo quando a velocidade na via é $\geq 50km/h$; o botão quadrado está ativo quando há congestionamento na via (Valor no Slider ≥ 8)

5 Avaliação de desempenho

(...)

6 Implementação real - refleções

(...)

7 Conclusão

(...)

Recursos