



UNIVERSIDADE FEDERAL DO CEARÁ

Projeto de Microprocessadores 2019.2 Projeto

Iury Lima Rosal (422067) // Vinícius Almeida (413129)
Luís Gustavo (418210) // Lucas Aragão (421601)

22 de Novembro 2019

1 Introdução

Nos dias atuais, o uso da tecnologia em vias auxilia no gerenciamento do tráfego e na manutenção das vias, tudo para a melhoria de fluxo de veículos e, ao mesmo tempo, provocando menos demora na entrega de mercadorias via uso de rodovias. O projeto visa o uso de sensores para fazer um controle do limite de velocidade na via e por identificar dentre duas vias aquela com melhor fluxo de carros. Tudo podendo ser monitorado e visualizado com o auxilio de um aplicativo em paralelo ao hardware com conexão bluetooth.

2 Materiais

- 1 Microcontrolador STM32F030F4P6 - Greenpill;
- 4 LEDs IR (Emissores);
- 4 Foto Diodos IR (Receptores);
- 4 resistores de 220 (para os LEDs IR (Emissores));
- 4 resistores de 330K (Para os foto diodos IR (Receptores));
- 1 módulo Bluetooth HC-05;
- 1 Programador/Gravador ST-LINK V2 STM8 e STM32 MCU;
- 3 Protoboards.

3 Funcionamento

A ideia geral do projeto é simular duas vias de trânsito de mesmo sentido. A partir disso, teremos sensores nessas vias que irão monitorar elas quanto ao trânsito. Os sensores juntamente com a greenpill irão fazer um cálculo e assim estimar qual via está mais congestionada e a velocidade de cada uma. O resultado desse processamento seria apresentado por um aplicativo, via bluetooth. Esse projeto seria muito útil em uma situação de emergência. Por exemplo, duas vias sendo que uma delas está cogestionada e um carro do corpo de bombeiros está indo atender uma emergência, ao saber da via mais rápida, ele evita o congestionamento e consegue atender de forma mais rápida o serviço. A imagem abaixo simula essa situação:



Figure 1: Simulador da situação para o projeto

Outra aplicação possível seria em uma via para carros autômatos, a partir do envio via bluetooth conjunto com um aplicativo. Todos os carros autômatos saberia a melhor opção a seguir, dessa forma, configurando um bom tráfego.

Os "blocos brancos" simbolizam os sensores infravermelhos que ajudarão a identificar o congestionamento e a calcular a velocidade de cada via. As protoboards simbolizam os canteiros laterais e o central. A ideia é que os sensores fiquem muito próximos entre si. Na situação real, a via teria esse par de sensores replicado várias vezes, conseguindo estimar de forma mais próxima o status da via.



Figure 2: Esquemático dos sensores

Seguindo o esquemático apresentado no simulador, realizamos a seguinte montagem do projeto:

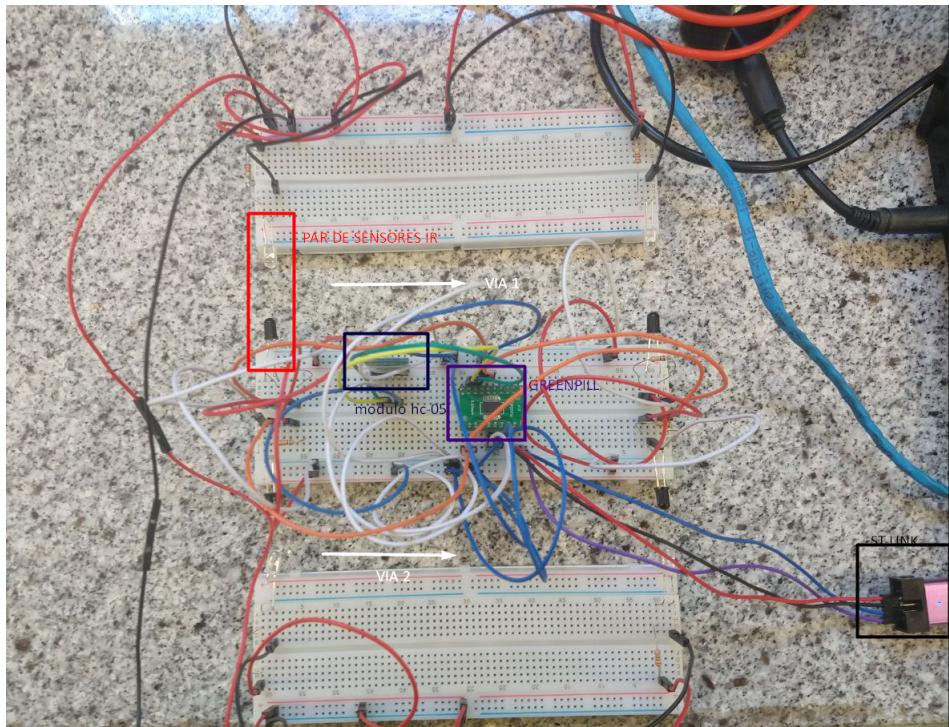


Figure 3: Montagem

CONFIGURAÇÃO NO CUBE

- Em relação ao ADCs, apenas "ativamos" as entradas PA0, PA1, PA5 e PA6. Não realizamos mais nenhuma configuração adicional.
- Em Connectivity, em USART1, ativamos o modo Asynchronous. Em Parameter Settings colocamos o Baud Rate em 96000.
- Em System Core, SYS, ativamos o Debug.
- Apesar do projeto ter Timer14 e Timer16 ativado. Utilizamos apenas o Timer14. Portanto, apenas ativamos o TIMER14. Em NVIC Settings, ativamos o modo interrupto. Em parameter settings, colocamos 79 em Prescaler e 99 em Counter Period.

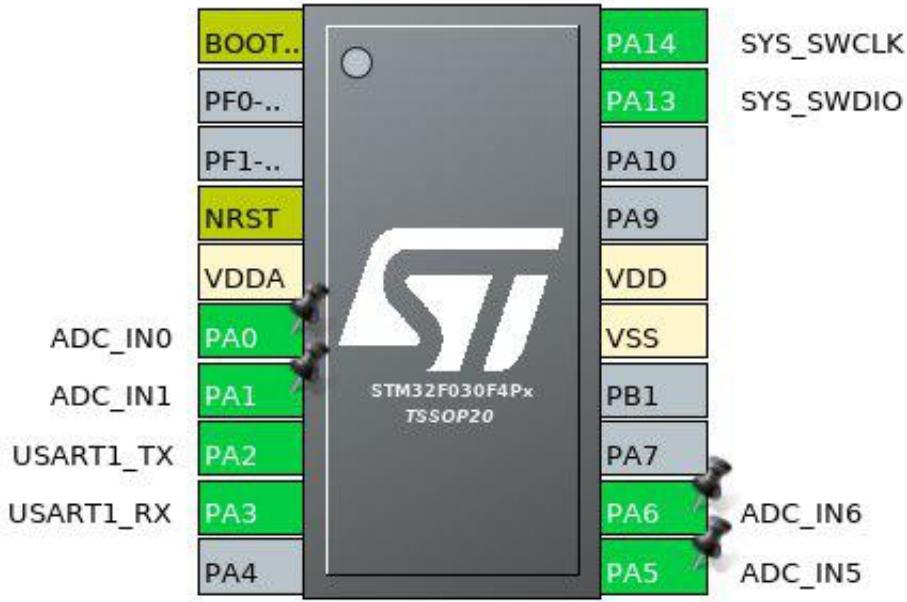


Figure 4: Montagem

CODIFICAÇÃO

Para o funcionamento do projeto, realizamos alguns tratos na codificação.

1. Função para trato de múltiplos canais ADC

Utilizamos 4 sensores no projeto (4 pares de sensores infravermelhos: 1 emissor e 1 receptor por par), então estamos trabalhando com 4 entradas ADC para receber a tensão de cada receptor IR. O próprio CUBE já configura esses canais no momento em que ativamos o uso deles (PA0, PA1, PA5 e PA6) na função static void MX-ADC-Init(void) na main.c. Como só podemos realizar uma leitura por vez (não tem como realizar a leitura das 4 entradas ADC simultaneamente, mas sim uma por vez), estruturamos uma função que permite selecionarmos qual entrada ADC desejamos realizar a leitura, realizando a configuração do canal desejado e, logo em seguida, obtendo a leitura.

A função a seguir recebe como parâmetro qual entrada ADC terá a leitura realizada. Se quisermos saber a leitura do PA0, por exemplo, que é o pino 5 da GreenPill, colocamos 0. A partir da escolha, um switch case irá determinar qual ADC-CHANNEL será configurado. Nossa projeto está configurado para as entradas ADC serem a PA0, PA1, PA5 e PA6, como já apresentado. Após a seleção do canal desejado, é realizada a configuração do canal selecionado, usando a função ADC-ConfigChannel. Após configurado o canal, realizamos a leitura do que está chegando nele pela função HAL-ADC-GetValue.

```

uint16_t readv(uint16_t number){
    ADC_ChannelConfTypeDef chConfig = {0};
    switch(number){
        case 0:
            chConfig.Channel = ADC_CHANNEL_0;
            break;
        case 1:
    }
}

```

```

        chConfig.Channel = ADC_CHANNEL_1;
        break;
    case 5:
        chConfig.Channel = ADC_CHANNEL_5;
        break;
    case 6:
        chConfig.Channel = ADC_CHANNEL_6;
        break;
    default:
        break;
}
HAL_ADC_ConfigChannel(&hadc, &chConfig);
HAL_ADC_Start(&hadc);
uint16_t v = HAL_ADC_GetValue(&hadc);
HAL_ADC_Stop(&hadc);
chConfig.Rank = ADC_RANK_NONE;
HAL_ADC_ConfigChannel(&hadc, &chConfig);
return v;
}

```

2. Função para encaminhar alguma string para o HC-05, via UART

Necessitamos enviar dados para o HC-05, para assim recebemos as informações no aplicativo desenvolvido no APP INVENTOR 2. Para isso, criamos uma função sendData que irá possibilitar o envio de caracteres da GreenPill para o HC-05, evitando a coleta de "lixo". INIT-CHAR E END-CHAR são variáveis definidas no começo do código (fora da int main) que irão delimitar o inicio e o fim do recebimento de dados. Um detalhe importante é que devido a esses dois caracteres, a "mensagem" que for colocada em um dos parâmetros da função deve ter 2 espaços a mais para receber esses dois caracteres. Por exemplo, char message[6] = "test". "test" só tem 4 caracteres, os outros 2 que sobram do tamanho total ($6 - 4 = 2$) são para receber o INIT-CHAR e o END-CHAR na função sendData. Em relação ao tamanho, é recomendado que para o parâmetro size utilize a função strlen da biblioteca <string.h>, retornando o tamanho da string.

```

/* Private define _____*/
#define INIT_CHAR "&"
#define END_CHAR "\n"

void sendData(u_int8_t *message, int size) {
    HAL_UART_Transmit(&huart1, (uint8_t *)
INIT_CHAR, sizeof(INIT_CHAR), 100);
    HAL_UART_Transmit(&huart1, (uint8_t *) message, size, 100);
    while(HAL_UART_GetState(&huart1) != HAL_UART_STATE_READY);
    HAL_UART_Transmit(&huart1, (uint8_t *) END_CHAR, sizeof(END_CHAR) +
sizeof(INIT_CHAR), 100);
}

```

O while dentro da função possibilita que o END-CHAR só seja inserido quando todos os caracteres da "message" forem "percorridos". A partir disso, é colocado o END-CHAR e o código é encerradom, evitando a coleta de "lixo". Outro ponto é que na main.c, na função static void MX-USART1-UART-Init(void), que é a função gerada pelo próprio CUBE para configurar a

UART, a configuração de BaudRate deve estar em 9600 para a função funcionar perfeitamente:
huart1.Init.BaudRate = 9600;

Com isso, usamos a função HAL-UART-Transmit para transmitir cada caractere pela conexão UART, especificamente a porta TX da Greenpill conectada com a porta RX do módulo HC-05. Estaremos utilizando o HC-05 apenas como Master, onde ele irá apenas receber dados da Greenpill e enviar para o aplicativo. A conexão da RX da Greenpill com a TX do módulo HC-05 é feita mesmo sem o seu uso, para evitar possíveis complicações.

3. Trato na conversão de int para string para "colocar" no sendData

Como pretendemos enviar os dados para o aplicativo, o envio, de modo mais prático, é via caracteres. Portanto pretendemos fazer os cálculos numéricos (velocidade e via mais rápida) pelo código da main.c e convertemos os resultados para uma única string que será trabalhada dentro do aplicativo. Isso ocorre pois o envio é serial, não tem como enviar várias strings simultaneamente de forma paralela. Para garantir que ambas as velocidades e a identificação da via mais rápida ocorram nos mesmo tempo, iremos "compactar" todos esses dados em uma única string. Para realizar a conversão de int para um char, basta realizamos os seguintes cálculos:

```
value0 = readv(0);
char msg[6] = {(value0/1000)%10 + 48,
                (value0/100)%10 + 48, (value0/10)%10 + 48,
                value0%10 + 48};
sendData(msg, strlen(msg));
HAL_Delay(200);
```

No exemplo acima, lemos o valor do ADC-CHANNEL-0, utilizando a função readv. Em seguida, pegamos 4 caracteres, pegando cada caractere por meio das divisões (/1000 pegará o primeiro caractere numa leitura da esquerda pra direita). Ao obter o caractere, é realizada uma divisão por 10, em que nos interessa apenas o resto, seguida de uma soma com 48. Na tabela ASCII, realizando essa operação em um valor inteiro você obtém o caractere correspondente ao valor. Realizando essas duas operações, conseguimos pegar cada "pedaço" do inteiro e converter em caractere, possibilitando o envio para o módulo HC-05 por meio da função sendData. Observe que msg tem tamanho 6 e só enviados 4 caracteres, pois deve existir espaço de 2 caracteres para o INIT-CHAR e o END-CHAR, como já foi mencionado.

4. Timer e CallBack

Utilizamos o TIM14 para utilizamos como relógio. Pois no cálculo da velocidade, teremos o espaço fixo e o tempo dependendo desse TIMER. O TIMER seguiu as seguintes configurações:

```
htim14.Instance = TIM14;
htim14.Init.Prescaler = 79;
htim14.Init.CounterMode = TIM_COUNTERMODE_UP;
htim14.Init.Period = 99;
```

O CallBack será a função que no fim de cada ciclo do timer irá executar alguma ação. Portanto podemos utilizar dentro dela variáveis globais e outras funções existentes, pois é como se fosse um "pedaço" da int main que será executada no final de cada ciclo. No caso das configurações do TIM14, o CallBack será chamado a cada 0.001 segundos, ou seja, 1 ms.

$$\frac{8000000Hz}{(79+1)(99+1)} = 1000Hz = \frac{1}{1000} = 0.001s = 1ms$$

A ideia é que quando o primeiro sensor de uma via for ativado, o contador comece marcando o tempo e que só seja finalizado quando o sensor subsequente seja ativado. Para isso utilizamos variáveis auxiliares que serão trabalhadas na int main.

```
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim14) {
    if (setTimerCount1){
        mSec1 = mSec1 + 1;
    }
    if (setTimerCount2){
        mSec2 = mSec2 + 1;
    }
}
```

setTimerCount é a variável auxiliar que servirá para saber quando começar e encerrar a contagem de tempo. Msec é a variável de contagem do tempo. O sufixo 1 e 2, se referem a via 1 (com os sensores referente ao PA0 e PA1) e a via 2 (com os sensores referente ao PA5 e PA6), respectivamente.

5. Comparação das velocidades

Fizemos uma simples função que resumidamente recebe dois parâmetros, compara qual valor numericamente é maior e retorna um char. Isso por conta que iremos enviar o char "1" para indicar que a via1 é mais rápida que a via2 ou iremos enviar o char "2" para o caso oposto. Esse caractere será adicionado a string que enviaremos ao HC-05 pelo sendData.

```
char fastest_road (uint16_t v1, uint16_t v2){
    char best_road;
    if (v1 > v2){
        best_road = '1';
    }
    else{
        best_road = '2';
    }

    return best_road;
}
```

6. Cálculo da Velocidade e envio pelo SendData

No while, iremos inicialmente verificar todos os sensores. Em caso do sensor coletar um valor acima de 2700 quer dizer que ele foi interrompido.

Verificamos se o primeiro sensor de cada via foi ativado (via1: sensor0 - PA0 | via2: sensor6 - PA6). Caso sim, zeramos o Msec e setamos o setTimerCount respectivo da via, ou seja, a contagem de tempo é iniciada naquela via para cálculo da via. A contagem do tempo ocorre devido ao CallBack, que é chamada a cada 1 ms, incrementando no Msec em caso de setagem do setTimerCount.

Identificado a ativação do primeiro sensor, é verificado o segundo sensor (via1: sensor1 - PA1

| via2: sensor5 - PA5). Se o segundo sensor for ativado, o setTimerCount da via respectiva é ressetado, ou seja, a contagem é encerrada. Em seguida, calculamos a velocidade. Estimamos que a distância entre cada sensor é de 15 cm. No código tratamos 15000 no cálculo por conta de float e com intuito para melhor demonstração. Se fossemos considerar o espaço de 15 cm, a velocidade na ativação e desativação dos sensores deveria ser muito rápida para a obtenção de resultados facilmente visíveis, o que compromete a viabilidade do projeto. Além disso, o trato com ponto flutuante poderia elevar o cálculo de uma forma mais complexa, também dificultando a demonstração do projeto. Logo após o cálculo da velocidade de cada via, é feita a comparação entre as velocidades, para ver qual via é a melhor opção.

Depois de todo esse processamento, é realizado o tratamento para envio da mensagem para o HC-05, que será posteriormente printada no aplicativo. Para isso realizamos as operações já apresentadas, colocando em uma única string a velocidade de cada via e o número da melhor via. Para separar cada dado usamos o "|", que será processado dentro do aplicativo. A atualização dessa string, consequentemente do envio da mensagem, só ocorre na ativação do segundo sensor em uma das vias.

Um ponto importante, digamos que o sensor0 é ativado e o sensor5 não é ativado. Nesse caso, a velocidade da via2 se manterá a anterior e será apenas atualizada a via1, já que nesse instante apenas a via1 está tendo algum tráfego. O mesmo vale para em caso do sensor5 ser ativado e o sensor0 não.

```
while (1)
{
    sensor0 = readv( (uint16_t) 0 );
    sensor1 = readv( (uint16_t) 1 );
    sensor6 = readv( (uint16_t) 6 );
    sensor5 = readv( (uint16_t) 5 );

    if(sensor0 > (uint16_t) 2700 ) {
        mSec1 = 0;
        setTimerCount1 = 1;
    }
    if(sensor1 > (uint16_t) 2700 ) {
        setTimerCount1 = 0;

        vel1 = (15000)/(mSec1);

        char choice = fastest_road(vel1 , vel2);

        char msg[9] = {(vel1/10)%10 + 48, (vel1)%10 + 48,
                      '|', (vel2/10)%10 + 48, (vel2)%10 + 48, '|',
                      choice };
        sendData(msg, strlen(msg));
        msg, strlen(msg), 100);
        HAL_Delay(500);
    }
    if(sensor6 > (uint16_t) 2700 ) {
        mSec2 = 0;
        setTimerCount2 = 1;
    }
}
```

```

if( sensor5 > ( uint16_t ) 2700 ) {
    setTimerCount2 = 0;

    vel2 = ( 15000 ) / ( mSec2 );

    char choice = fastest_road( vel1 , vel2 );

    char msg[9] = { ( vel1 / 10 ) % 10 + 48 , ( vel1 ) % 10 + 48 ,
        '|', ( vel2 / 10 ) % 10 + 48 , ( vel2 ) % 10 + 48 , '|', choice };
    sendData( msg , strlen( msg ) );
}
}

```

APP INVENTOR 2

Aqui apresentaremos como o projeto ficou estruturado no aplicativo desenvolvido na plataforma online APP INVENTOR 2. A ideia do aplicativo é apenas conectar via bluetooth, receber os dados da Greenpill com auxílio do módulo HC-05 e "printá-los" no aplicativo para o usuário.

1. Conexão via Bluetooth

Colocamos um ListPicker, um BluetoothClient (ambos auxiliam na conexão do aplicativo com o HC-05) e um Notifier (que irá realizar a comunicação com o usuário, mostrando avisos). Isso juntamente com dois botões (um de Conectar e outro de Desconectar).

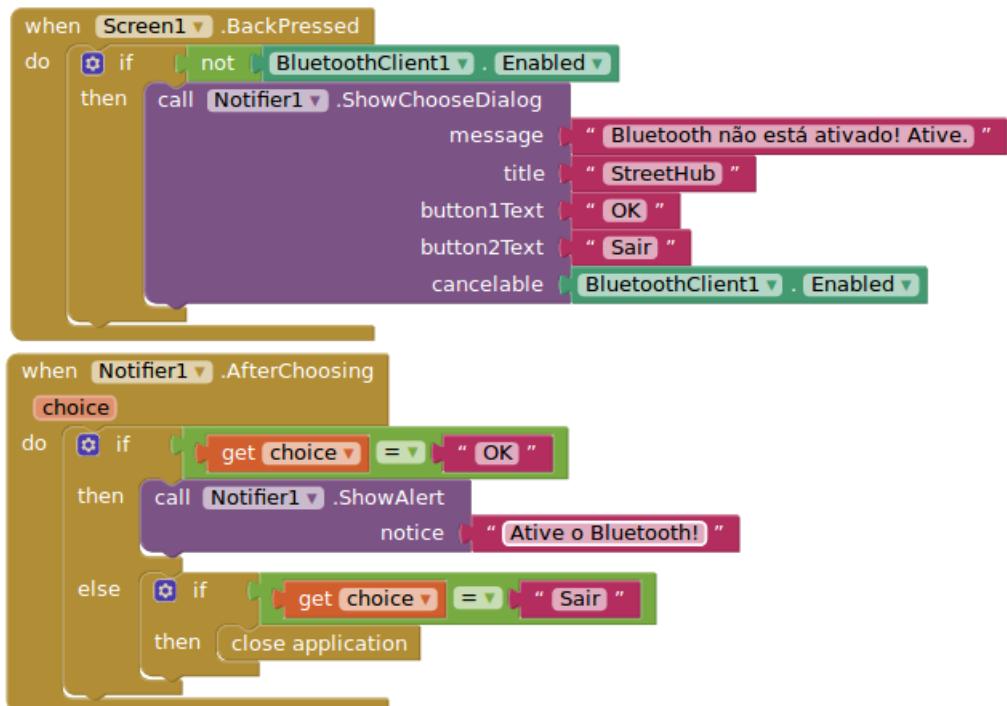


Figure 5: Esquemático do Notifier com BluetoothClient

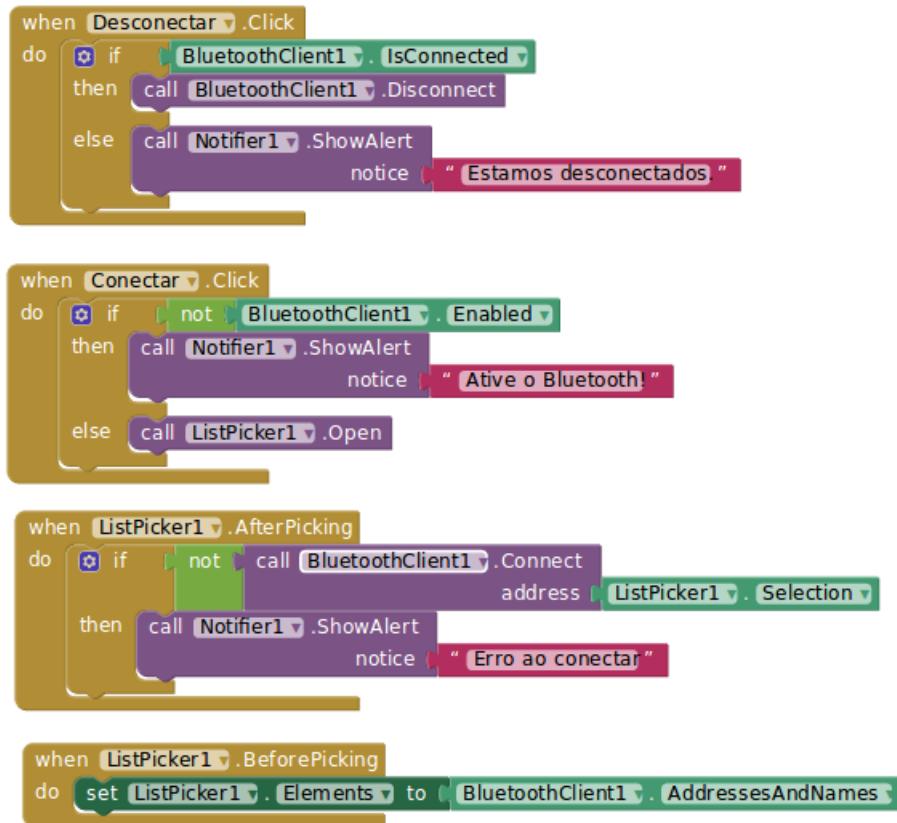


Figure 6: Esquemático da exibição das opções de conexão do Bluetooth com auxílio do ListPicker e BluetoothClient, juntamente com conexão

2. Processamento da string enviada pelo sendData

Com bluetooth conectado e enviando algo (diferente de 0), recebemos a string enviada para o hc-05. Logo em seguida, repartimos a string em "pedaços" tendo como referências os "|". Cada pedaço respectivo é colocado no seu devido lugar a ser printado. Tivemos um cuidado com o "e comercial" e o "/n" adicionados pela função sendData.

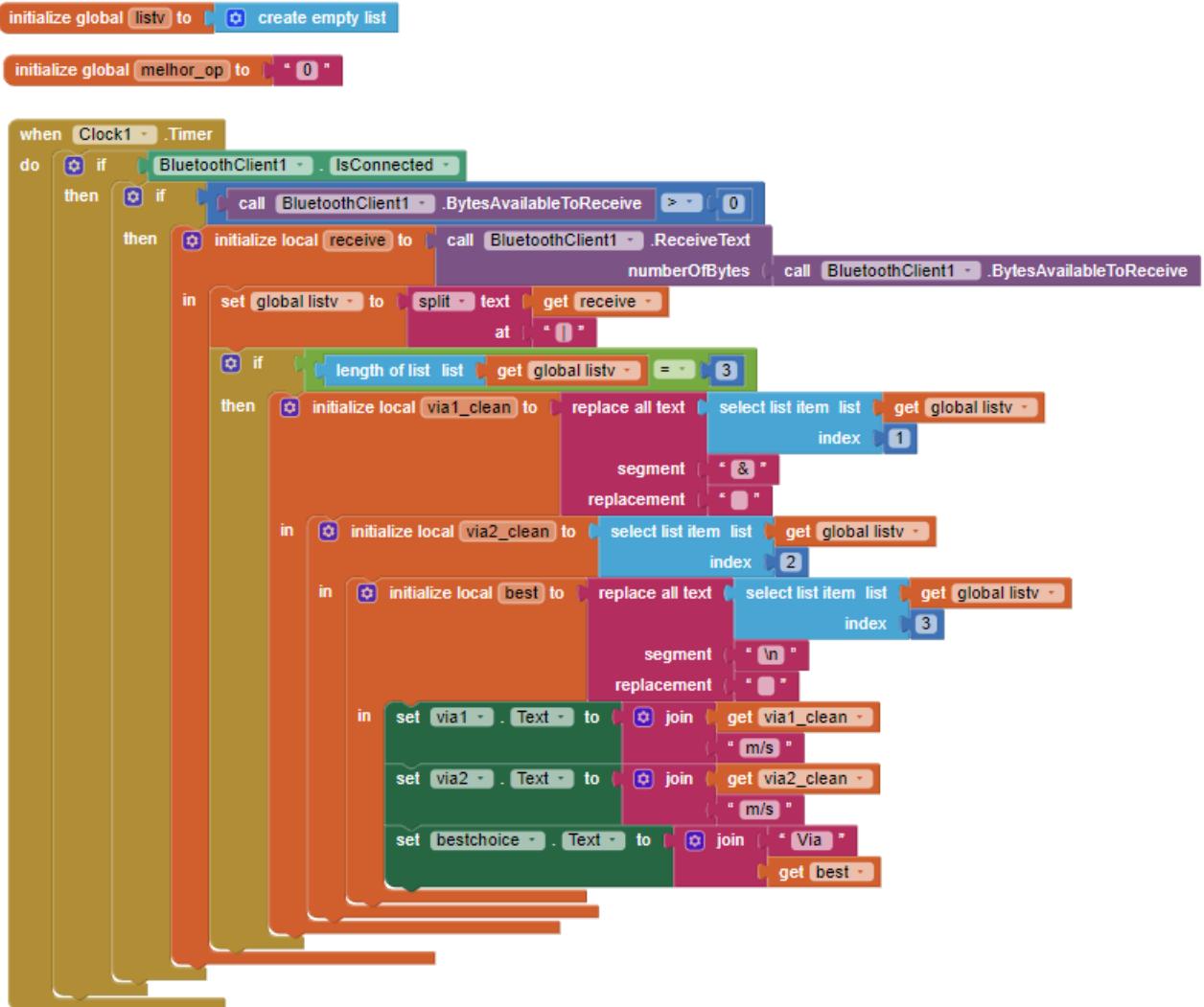


Figure 7: Recebimento e Processamento dos Dados no App

4 Diagrama de Blocos

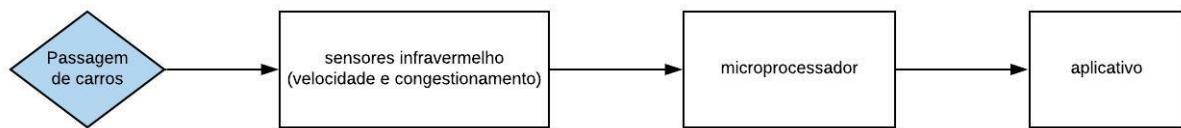


Figure 8: Diagrama de blocos indicando as interações entre as partes dos projetos

5 Cronograma

Table 1: Cronograma antes da Execução

14/10 - 21/10	Semana 1	Planejamento e Documentação do Projeto
21/10 - 28/10	Semana 2	Compra dos Materiais e Estudo da Codificação
28/10 - 04/11	Semana 3	Teste dos Materiais e Desenvolvimento do Código
04/11 - 11/11	Semana 4	Início de montagem e testes
11/11 - 18/11	Semana 5	Montagem e Ajustes
18/11 - 02/12	Semana 6 e 7	Testes, Ajustes e Entrega

O cronograma pode sofrer mudanças no decorrer da realização do projeto.

Table 2: Relatório

04/11	14h-16h	Teste do ST-LINK e do Software
05/11	8h-10h	Início dos testes dos sensores
08/11	8h-11h	Teste dos Sensores e começo do teste com HC-05
11/11	14h-16h	Teste com HC-05
12/11	7h30-10h e 14h-16h	Início da montagem do projeto - Sensores e Módulo HC-05
19/11	7h30-11h e 12h30-14h	Testes e Estudo do Timer
20/11	14h-16h	Testes e Estudo do Timer
21/11	7h30-11h e 12h30-15h	Testes com Timer e começo do cálculo da velocidade
22/11	8h-11h30 e 12h30-14h30	Finalização e Teste do Aplicativo
22/11	16h-18h	Atualização da documentação

6 Repositorio no GitHub

<https://github.com/viniciusAC/Estrada-Inteligente>