Processamento de Linguagens (3o Ano de Mestrado Integrado em Engenharia Informática)

Trabalho Prático 2 - Processamento de trilhos GPS

Relatório de Desenvolvimento

Diogo André Teles Fernandes (A78867)

João Miguel Freitas Palmeira (A73864)

Luís Manuel Meruje Ferreira (A78607)

7 de Maio de 2018

Resumo

Este trabalho prático tem como principais objetivos aprofundar e aplicar conhecimentos obtidos durante as aulas teóricas e práticas de Processamento de Linguagens. Além de pretender aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases e de, através destas, desenvolver Processadores de Linguagens Regulares que filtrem ou transformem textos, pretende também utilizar o sistema de produção para filtragem de texto em *C* FLEX.
Aplicando o mesmo método de escolha do primeiro trabalho, selecionamos o último enunciado disponível para este segundo trabalho prático, mais concretamente, o referente ao Processamento de Trilhos GPS.
Os resultados obtidos estão de acordo com os objetivos previamente definidos e com o que foi pedido no enunciado.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Processamento de Trilhos GPS

Área: Processamento de Linguagens

Um ficheiro *GPX* é um esquema *XML* que pode ser usado para descrever pontos de passagem (*waypoints*), um ou mais trilhos de GPS ou ainda rotas (*routes*). Para além deste, existem também ficheiros do tipo *KML* que servem um propósito semelhante, permitindo definir trilhos e pontos, entre outros. No entanto, estes últimos são direcionados para exibir dados geográficos em navegadores *Earth* tais como o *Google Earth* e o *Google Maps*. À semelhança de qualquer *GPX*, o *KML* utiliza uma estrutura de *tags* com elementos e atributos aninhados e baseia-se no padrão *XML*.
O enunciado que nos foi atribuído descreve a implementação de um filtro FLEX que transforma um documento relativo a um trilho em formato *GPX*, num documento em formato *KML*.
Para a implementação deste projeto optamos por usar a linguagem C++ com o FLEX, em vez de C. Consideramos que a linguagem C++ nos disponibiliza estruturas de dados adicionais, não presentes em C, que facilitam a resolução do problema.
Para além disso, apresentamos duas alternativas de resolução. A primeira descreve um algoritmo mais simples, que apenas faz tradução de trilhos, desenhando uma linha com esse trilho no mapa e apontando um ponto no mapa a cada 15 que surgem na descrição do trilho. A segunda versão é um pouco mais elaborada. Os trilhos são formatados como sendo *tracks*, e são calculados alguns detalhes sobre os mesmos, por exemplo o comprimento total do trilho, altitude máxima e mínima ao longo do trajeto, entre outros.

Estrutura do Relatório

Este relatório está organizado em cinco capítulos, seguidos de anexos, sendo que nestes últimos é apresentado todo o código desenvolvido. No início do documento existe também um resumo do mesmo.

No capítulo 2, é inicialmente feita uma descrição informal do problema proposto, traçando as linhas gerais do mesmo e de seguida são especificados os requisitos concretos que devem ser cumpridos.
De seguida, o capítulo 3 aborda todas as opções tomadas ao longo da realização do trabalho prático, assim como as decisões que lideraram o desenho da solução e da sua implementação.
No quarto capítulo é explicado o porquê de termos optado por abordar o problema do modo que abordamos explicando as dificuldades que enfrentamos durante a resolução da mesmo. São apresentados também os resultados de alguns testes realizados.
Por fim, o capítulo 5 inclui uma síntese do trabalho realizado com uma análise crítica dos resultados obtidos, possíveis melhorias e ainda trabalho futuro que poderia ser desenvolvido.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

2.1.1 Desenvolvimento em FLEX de um filtro que transforme um documento em formato GPX no formato *KML*

Neste trabalho estão envolvidos dois tipos de ficheiros, tal como referido anteriormente: *GPX* e *KML*.

Pretende-se que sejam recolhidos alguns ficheiro do tipo *GPX* com o intuito de se realizarem experiências ou testes a partir destes, e pretende-se também que seja desenvolvido um processador de texto FLEX que os transforme em ficheiros do formato *KML*. Os elementos do formato *GPX* que foram considerados para tradução foram os trilhos GPS (toda a informação associada a eles) e os pontos de passagem (*waypoints*).

2.2 Especificação dos Requisitos

Os requisitos fundamentais deste enunciado são:

- Recolher todos os campos sobre um trilhos que estejam presentes num ficheiro *GPX*;
- Recolher toda a informação sobre todos os pontos de passagem de cada trilho;
- Gerar um ficheiro *KML* com toda a informação recolhida anteriormente, ou seja, relativa aos trilhos e aos pontos de passagem do trilho;

Capítulo 3

Conceção/desenho da Resolução

3.1 Estruturas de Dados

3.1.1 Desenvolvimento em FLEX de um filtro que transforme um documento em formato *GPX* no formato *KML*

Quanto a este tópico podemos afirmar que nas duas versões realizadas estão presentes duas estruturas de dados: uma estrutura para definir os pontos e outra com os vetores formados com esses mesmos pontos.
A estrutura dos pontos é composta pelas suas coordenadas (latitude e longitude) bem como por um inteiro que servirá de referência para identificar cada ponto.

```
double lat;
double lon;
int pointNr = 0;
```

Já a estrutura dos vetores é composta pelas coordenadas de cada vetor.

```
vector<double> lons, lats;
```

3.1.2 Versão alternativa

Comparativamente à versão anterior, foram introduzidas novas variáveis à estrutura de dados anteriormente elaborada. Foram acrescentadas as seguintes variáveis:

- double:
  - prevLat
  - prevLon
  - dlon
  - dlat
- int:
  - noTime
  - speedValuesCount
- char:
  - time
- float:
  - speedsSum

spped

maxAltitude

minAltitude

altitude

distance

a

Esta nova estrutura permite-nos guardar nova informação sobre o trilho, ou seja, permite-nos guardar informação do tipo: diferentes altitudes, altitude máxima e mínima atingidas; distância percorrida; tempo total do percurso; velocidade alcançada bem como as somas das diferentes velocidades atingidas. Será possível então guardar a informação mais detalhada de cada percurso. Além disso, foi ainda acrescentado um vetor extra, que nos permite guardar os tempos.

3.2 Algoritmos

3.2.1 Desenvolvimento em FLEX de um filtro que transforme um documento em formato *GPX* no formato *KML*

Incialmente começou-se por escrever três *MACROS*:

```
#define HEADER
#define SETTINGS
#define MARKER
```

Cada um deles descreve uma string que contém a estrutura XML para uma dada secção do ficheiro KML. O *HEADER* define a abertura do documento. O SETTINGS define um estilo para a linha que mostra o trilho no mapa. Finalmente o MARKER, que contém os elementos necessários para representar um marcador (ou ponto). Cada um destes MACROS contém um conjunto de elementos, que apresentamos a seguir com uma breve explicação dos mesmos:

- **LineStyle**: Delimita a zona que define o estilo de desenho para toda a geometria da linha do trilho.
- **Color**: Especifica a cor que iremos atribuir ao desenho da linha do trilho.
- **Width**: Especifica a largura da linha, em pixeis.
- **Point**: Especifica uma localização geográfica definida por longitude, latitude e altitude (opcional).
- **LineString**: Especifica um conjunto conectado de segmentos de linha (o trilho).
- **Extrude**: Especifica se deve conectar o ponto ao solo com uma linha (é um valor boleano).
- **Tessellate**: Especifica se deve permitir que a *LinearString* siga o terreno (valor boleano).
- **AltitudeMode**: Especifica como os componentes de altitude no elemento *¡coordinates¿* são interpretados. No nosso caso estes são ignorados e é utilizada a altura do terreno.
- **Coordinates**: Especifica dois ou mais tuplos de coordenadas, cada uma consistindo em valores de ponto flutuante para longitude, latitude e altitude.

Adicionou-se ainda uma estrutura de vetores que já foi explicada anteriormente. De seguida definimos, na primeira parte do ficheiro FLEX, algumas *start conditions* tais como a *IGNORE* e *PATH* para utilizar no processamento do filtro. Atribuímos também nomes a alguns padrões lexicais (*trkBegin* e *trkEnd*), de maneira a melhorar a estrutura do nosso código.

Já na segunda secção do ficheiro são declaradas as variáveis a utilizar, como explicado anteriormente.
Quando se faz *parsing* do *<gpx* faz-se *printf* do *HEADER* e é iniciada a *start condition IGNORE*. Quando esta *start*

condition está ativa e o flex encontra a informação contida no *trkBegin*, é realizado um *printf* do *SETTINGS* e é iniciada a *start condition PATH*. Por outro lado, quando se encontra qualquer outra coisa ou uma mudança de linha (n), não se faz nada. Quando a *start condition PATH* está ativa e é encontrado o *trkEnd* são imprimidos os fechos das *tags: coordinates, LineString e Placemark*.

De outro modo, caso seja encontrado *<trk* é realizado uma recolha das coordenadas (latitude e longitude) através de um *sscanf* e, posteriormente, essa informação recolhida é imprimida e guardada também na estrutura. Caso o número daquele ponto seja zero ou múltiplo de quinze, o mesmo é também introduzido nos vetores criados, para posteriormente a se escrever todo o trilho, esses pontos serem escritos como marcadores. Finalmente, é incrementado um ao número de pontos que foram percorridos.

Quando o ficheiro chega ao fim (*End of File*), é verificado se o número do ponto final é múltiplo de quinze e, caso não seja, irá realizar *push_back* à informação do mesmo (último ponto). Caso contrário retorna-se zero (não se faz nada). Isto é feito para guardar o ponto final do trilho, caso esse ainda não tenha sido colocado nos vetores.

Por último, definimos a função *main* na qual começamos por delinear um iterador para os vetores bem como definindo um inteiro para a posição de cada um dos vetores. Prontamente são fixados dois ficheiros, um de leitura e outro de escrita, que serão recebidos como argumentos da função *main*. Ainda dentro da função *main*, é invocada a função *yylex* e é realizado um ciclo *for* onde enquanto existirem pontos nos vetores que foram construídos, o *MARKER* é imprimido com a informação relativa a cada ponto. Acabando o ciclo *for*, é realizada a última impressão no documento *KML* fechando as tags *Document* e *kml*.

3.2.2 Versão alternativa

Nesta versão alternativa foram feitos cinco defines:

```
#define HEADER
#define COORD
#define MARKER
#define WHEN
#define EARTH_RADIUS
```

O define COORD será usado para introduzir coordenadas na *track*, invés de ser um conjunto de pontos, o WHEN para introduzir os tempos e por último o *EARTH_RADIUS* que define o raio da terra (6371), valor usado nos cálculos. Além disso foi criado um vetor extra, para guardar os tempos.

Aqui, passamos a definir o trilho como sendo um elemento *gx:track* e não como um conjunto conectado de segmentos de linha(*LineString*).
Tal como na versão anterior, na primeira secção do ficheiro FLEX definimos as mesmas *start conditions: IGNORE e PATH*. Embora, para além dos padrões lexicais definidos antes (*trkBegin* e *trkEnd*), e com o mesmo intuito, definimos os seguintes padrões:

- *trkPtTime*
- *trkPtBegin*
- *trkPtEnd*
- *trkPtSpeed*
- *trkPtEle*

Na secção seguinte, e comparativamente à versão inicial, existiu uma adaptação da estrutura de dados como foi mencionado anteriormente.

Seguidamente, de modo similar à primeira versão, quando se encontra o *gpx* é imprimido o *HEADER*. No entanto, além disso é invocada a função *printStyles* (explicada mais à frente) e é iniciada a *start condition IGNORE*. Quando esta *start condition* está ativa e se encontra o *trkBegin* é imprimida a informação sobre o nome da *track* e iniciada a *start condition PATH*. Caso contrário não realiza nada.

Posteriormente, com a *PATH* iniciada, no momento em que se encontre o *trkPtBegin*, são obtidos os valores da latitude e longitude, inseridos nos vetores respetivos e é incrementado o *point number*. A seguir realiza-se uma conversão destas coordenadas para radianos e é feito o cálculo da distância entre o ponto atual e o anterior (se existir) segundo as informações contidas no *website* [1]. Essa distância é depois adicionada à distância total do trilho.

Ainda na mesma *start condition*, quando se encontrar o padrão lexical *trkPtTime* cria-se uma variável para colocar o tempo que vamos adicionar e recolhe-se a informação sobre tempo. É de seguida guardado uma duplicação desse tempo na variável com um *strdup* e feito um *pushback* para o vetor *times* desse tempo guardado . É ainda imprimido o valor do tempo, recorrendo à MACRO *WHEN*. Finalmente, define-se a variável *noTime* como sendo zero. A variável noTime permite garantir que existe um valor de tempo para cada coordenada, pois ao usar *gx:track*, não podem existir menos valores temporais do que coordenadas.

Decidimos adiar a impressão das coordenadas para o fim do trilho, para seguir o formato usado nos exemplos fornecidos pela google, em que todos os itens de tempos surgem antes das coordenadas.
Caso o padrão lexical seja o *tkrPtSpeed* recolhe-se a informação relativa à velocidade, soma-se à variável que guarda o total das velocidades e incrementa-se o contador de velocidades guardadas, para no fim ser calculada uma média.

Já no caso do *tkrPtEle*, é recolhida a informação sobre a altitude e verificada se a altitude em questão é a maior ou a menor em relação ao percurso até então obtido e analisado. Se esta for maior que o valor da maior altitude registada é então guardado como a maior altitude ou então se for a menor altitude é registada como a menor altitude encontrada.

Quando é encontrado o padrão lexical *trkPtEnd*, se a variável *noTime* for zero passa essa variável a um, caso contrário avisa que há informação em falta no ficheiro.

Por último, quando se encontrar o padrão *trkEnd* corre um ciclo *for* em todos os pontos e imprime a latitude e longitude dos mesmos com o *COORD* criado no *define*. Após isso é fechada a *tag gx:Track* e aberta a *tag description*. É inserida a distância calculada anteriormente e são imprimidas todas as estatísticas calculadas ou obtidas ao longo do percurso (tempo de início e fim, velocidade média, altitude máxima e mínima). De seguida fecha-se a *tag description* e abre-se a *tag Placemark* onde será colocado o marcador de início a verde e o do fim a vermelho. Por fim, fecham-se as tags de fim de *document* e de *kml*.

A função *printStyles* é responsável por definir vários estilos que vão ser utilizados no *KML*:

- Ícone verde normal
- Ícone verde selecionado
- Ícone verde
- Ícone vermelho normal
- Ícone vermelho selecionado
- Ícone vermelho
- Linha do trajeto

Finalmente, temos a função *main* onde (tal como na versão anterior), são inicializados os dois ficheiros, de leitura e de escrita, que vão ser passados como argumentos e definidos como sendo os input e output do FLEX.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Na solução do problema imposto, decidimos resolver o problema por duas alternativas. A primeira é uma implementação simples que apenas mostra, quando visualizado no *My Maps*, o caminho representado no ficheiro *GPX* introduzido e *Placemarks* de 15 em 15 coordenadas.

A segunda foi uma melhoria em relação à primeira, onde além de ser mostrado o caminho, existem também detalhes sobre a distância do percurso, e várias estatísticas.

Nesta segunda implementação, decidimos utilizar *gx:track* em vez do *Linestring* usado na primeira versão, por ter características que poderão ser úteis para funcionalidade futura. De destacar a capacidade deste modelo de representação de dados de representar valores nulos para quaisquer elementos exeto valores temporais. Esses valores são depois interpolados a partir dos dois pontos vizinhos mais próximos que estejam bem definidos nesses elementos.
Inicialmente a dificuldade foi perceber como aproveitar as *start conditions* para conseguirmos obter a estrutura desejada no ficheiro *KML*. Posteriormente, foi-nos complicado o uso do *sscanf* para tentar tirar strings entre *tags*, visto que apanhava também a tag de fecho. Vários problemas de implementação foram solucionados quando consultamos o manual do flex [2].

4.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos e os respetivos resultados obtidos:

4.2.1 Ficheiro de Exemplo Utilizado nos Testes

Decidimos usar este ficheiro para demonstrar os resultados obtidos no relatório visto que é dos exemplos mais reduzidos que conseguimos obter. Deste podemos verificar como é a estrutura base de um ficheiro *GPX*.

```
<?xml version='1.0' encoding='UTF-8'?>
<gpx version="1.1" creator="JOSM GPX export" xmlns="http://www.topografix.com/GPX/1/1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd">
  <metadata>
    <bounds minlat="45.489266" minlon="9.638791" maxlat="45.4937886" maxlon="9.647482"/>
  </metadata>
  <trk>
    <trkseg>
      <trkpt lat="45.49372845305671" lon="9.646934861540794">
```

```
      </trkpt>
      <trkpt lat="45.49378861924987" lon="9.647272819876669">
      </trkpt>
      <trkpt lat="45.493626922459974" lon="9.647331828474998">
      </trkpt>
      <trkpt lat="45.49336745454783" lon="9.647396201491356">
      </trkpt>
      <trkpt lat="45.49293124492887" lon="9.647482032179836">
      </trkpt>
      <trkpt lat="45.49278834793972" lon="9.647337192893028">
      </trkpt>
      <trkpt lat="45.49261160642523" lon="9.64710115849972">
      </trkpt>
      <trkpt lat="45.49239725958898" lon="9.64678465783596">
      </trkpt>
     ...
      </trkpt>
    </trkseg>
  </trk>
</gpx>
```

4.2.2 Desenvolvimento em Flex de um filtro que transforme um documento em formato GPX no formato KML

Com a utilização da primeira versão feita para o filtro, ou seja com o *gps-1.lex*, no ficheiro mostrado no capítulo anterior, foi obtido o seguinte ficheiro *KML*, que posteriormente foi importado no *My Maps* da *Google* e obtida a imagem da figura 4.1.

```
<?xml version='1.0' encoding='UTF-8'?>
<kml xmlns="http://www.opengis.net/kml/2.2"> <Document><name>Caminho</name><description>Descricao de Teste</d
        <Style id="LinhaAmarela">
                <LineStyle>
                        <color>7f00ffff</color>
                        <width>20</width>
                </LineStyle>
                <PolyStyle>
                        <color>7f00ff00</color>
                        <width>2</width>
                </PolyStyle>
        </Style>
        <Placemark>
        <styleUrl>#LinhaAmarela</styleUrl>
        <LineString>
        <extrude>1</extrude>
        <tesselate>1</tesselate>
        <altitudeMode>clampToGround</altitudeMode>
        <coordinates>
9.646935, 45.493728,0
9.647273, 45.493789,0
9.647332, 45.493627,0
...
9.641993, 45.489306,0
9.642036, 45.489266,0
</coordinates>
```

```
</LineString>
</Placemark><Placemark>
        <Point>
                <coordinates>9.646935,45.493728</coordinates>
        </Point>
</Placemark>
<Placemark>
        <Point>
                <coordinates>9.646102,45.491955</coordinates>
        </Point>
</Placemark>
...
<Placemark>
        <Point>
                <coordinates>9.642036,45.489266</coordinates>
        </Point>
</Placemark>
</Document></kml>
```

4.2.3 Execução da primeira versão

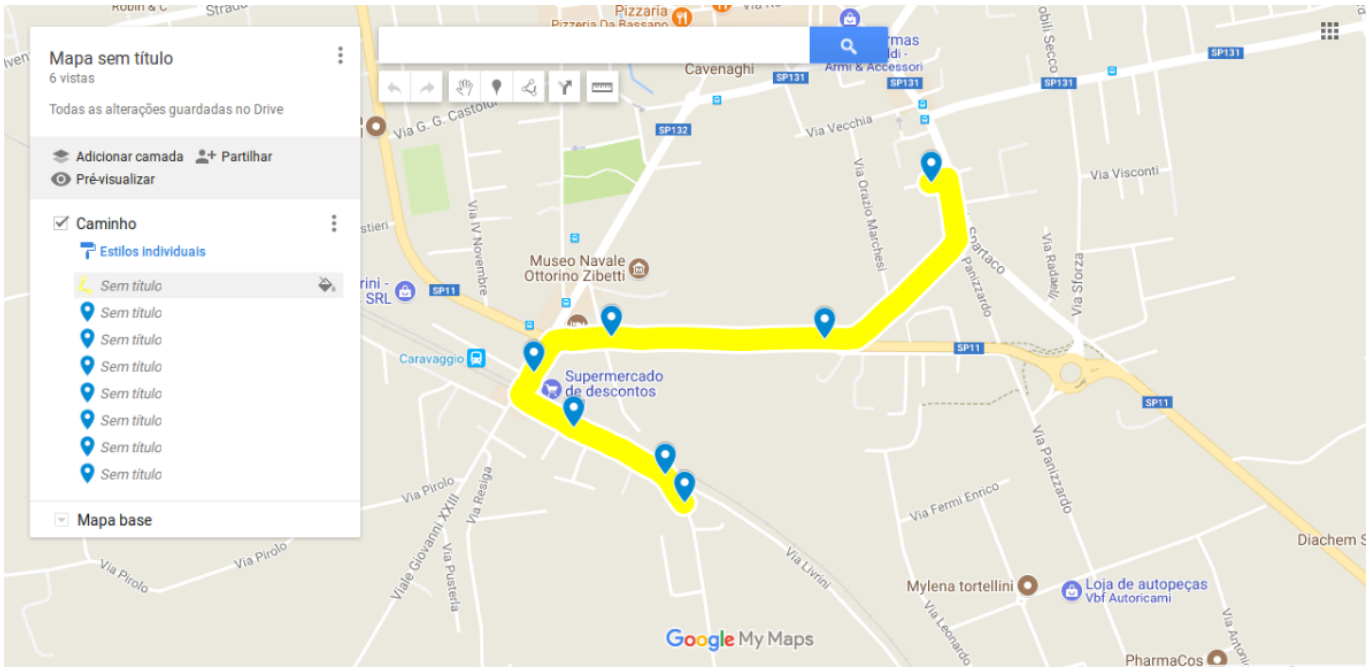

Figura 4.1: Exemplo da execução do ficheiro *KML* gerado com o gps1 no *My Maps*

4.2.4 Execução da versão alternativa

Da execução da versão alternativa, com o ficheiro em 4.2.1, obtemos as seguintes imagens. A primeira (4.2) demonstra o trilho obtido, quando importado no *My Maps*. A segunda mostra as estatísticas quando se carrega na opção **Pré-visualizar** visível na primeira imagem e se carrega em algum ponto do percurso. Tal como se pode verificar, é possível ver a distância aproximada calculada, os *timestamps* de inicio e fim, a velocidade média, altitude máxima e mínima, quando estas são possíveis obter.

Neste exemplo as mesmas não davam para obter, mas no outro exemplo enviado, em conjunto com o trabalho, as mesmas já estarão disponíveis.

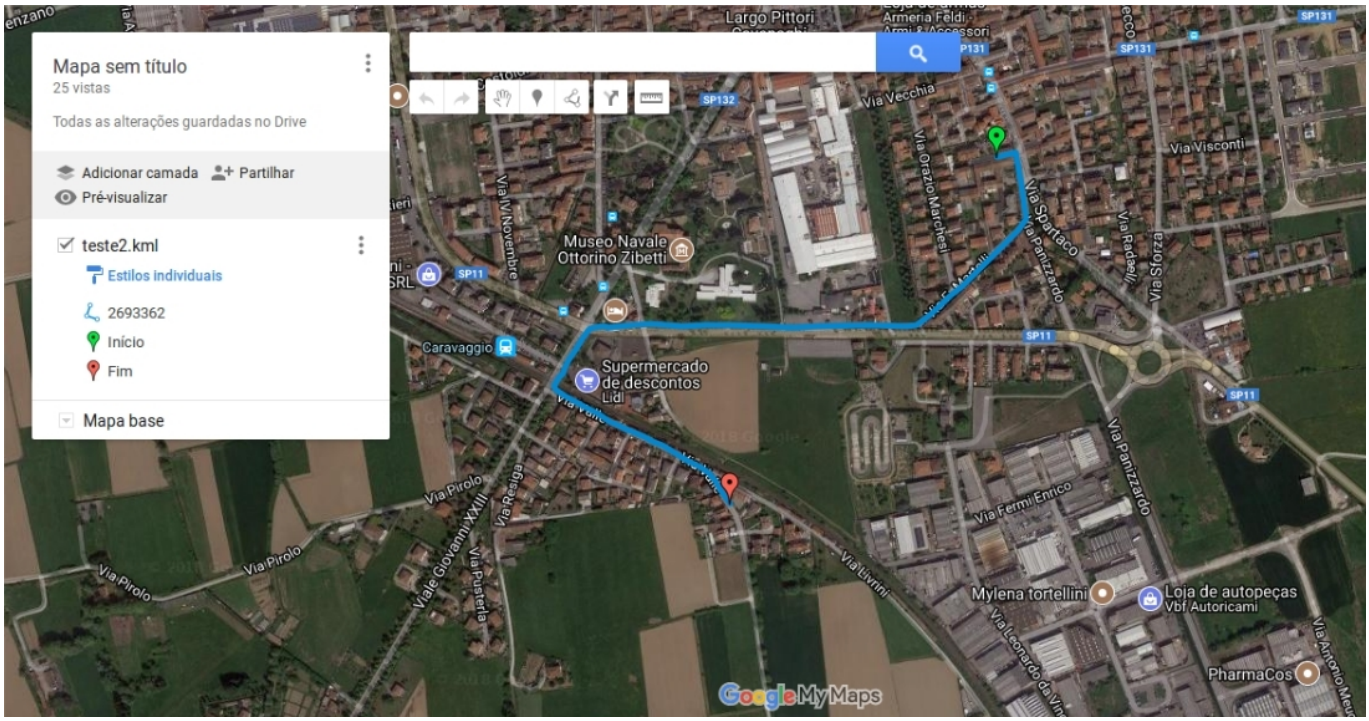

Figura 4.2: Exemplo da execução do ficheiro *KML* gerado no gps2 no *My Maps*

Figura 4.3: Exemplo da execução do ficheiro *KML* gerado no gps2 no *My Maps*

Capítulo 5

Conclusão

Em suma, este projeto permitiu-nos aprofundar os nossos conhecimentos na área de Processamento de Linguagens em relação aos tópicos lecionados durante as aulas, tendo desta vez o foco sido direcionado para a ferramenta de processamento de texto FLEX.

Desenvolvemos duas versões alternativas para a resolução do problema, uma mais simples e outra mais elaborada. Na segunda foi até possível elaborar algumas estatíticas sobre os percursos analisados.
Julgamos ter conseguido cumprir os objetivos definidos no enunciado de forma satisfatória.

Como sugestões para trabalho futuro, poderiam ser tidos em conta mais elementos de informação para formular estatísticas na alternativa mais elaborada. Por exemplo, poderia ter-se em conta também a possibilidade de registar níveis de batimento cardíaco, quando essa informação estivesse disponível. Para além disso, aqui também poderiamos ter representado os pontos de passagem (waypoints) contidos no ficheiro gpx, que por só termos notado a sua existência muito próximo do prazo de entrega, não foi possível implementar.

Apêndice A

Código do Programa

Mostra-se a seguir o código desenvolvido para este trabalho prático e que está explicado na secção de Algoritmos do capítulo 3.

A.0.1 Filtro em Flex que transforma um documento *GPX* num documento *KML*

```
%{
 #include <vector>
 #define HEADER "<kml xmlns=\"http://www.opengis.net/kml/2.2\"> <Document><name>Caminho</name><description>
 Descricao de Teste</description>\n\t<Style id=\"LinhaAmarela\">\n\t\t<LineStyle>\n\t\t\t<color>7f00ffff
 </color>\n\t\t\t<width>20</width>\n\t\t</LineStyle>\n\t</Style>\n"
 #define SETTINGS "\t<Placemark>\n\t<styleUrl>#LinhaAmarela</styleUrl>\n\t<LineString>\n\t<extrude>1
 </extrude>\n\t<tesselate>1</tesselate>\n\t<altitudeMode>clampToGround</altitudeMode>\n\t<coordinates>\n"
 #define MARKER "<Placemark>\n\t<Point>\n\t\t<coordinates>%lf,%lf</coordinates>\n\t</Point>\n</Placemark>\n"
 using namespace std;
 vector<double> lons, lats;
%}
%option noyywrap outfile="gps.cc"
%s IGNORE PATH
trkBegin  .*\<trk\>.*
trkEnd  .*\<\/trk\>.*


%%
  double lat;
  double lon;
  int pointNr = 0;

\<gpx.* {printf(HEADER); BEGIN IGNORE;}
<IGNORE>{trkBegin} {printf(SETTINGS); BEGIN PATH; }
<IGNORE>.|\n {}
<PATH>.|\n {}
<PATH>{trkEnd}  {printf("</coordinates>\n</LineString>\n</Placemark>");}
<PATH>\<trkpt.*  {
                  sscanf(yytext,"<trkpt lat=\"%lf\" lon=\"%lf\">",&lat, &lon);
                  printf("%lf, %lf,0\n",lon,lat);
                  if(pointNr == 0 || pointNr % 15 == 0){
                    lons.push_back(lon);
                    lats.push_back(lat);
                  }
```

```
                pointNr++;
              }

<<EOF>> {
          if(pointNr % 15 != 0){
            lons.push_back(lon);
            lats.push_back(lat);
          }
          return 0;
        }
%%
int main(int argc, const char* argv[]) {
  vector<double>::iterator itLons, itLats;
  int position;
  FILE *fw;
  if(argc>1){
    yyin=fopen(argv[1],"r");
  }
  if(argc>2){
    fw =fopen(argv[2],"w");
    dup2(fileno(fw),1);
  }



  yylex();
  for(position = 0, itLons = lons.begin(),itLats = lats.begin();itLats != lats.end() && itLons != lons.end();
  itLons++,itLats++,position++){
 printf(MARKER,lons.at(position),lats.at(position));
  }
  printf("</Document></kml>");
  return 0;
}
```

A.0.2 Versão alternativa

```
%{
  #include <vector>
  #include <math.h>
  #define HEADER "<kml xmlns=\"http://www.opengis.net/kml/2.2\">\n<Document>\n"
  #define COORD "\t\t\t<gx:coord>%lf %lf %lf</gx:coord>\n"
  #define MARKER "\t<Placemark>\n\t\t<name>%s</name>\n\t\t<description><![CDATA[Latitude: %lf<br>Longitude:
  %lf]]></description>\n\t\t<styleUrl>%s</styleUrl>\n\t\t<Point>\n\t\t\t<coordinates>%lf,%lf</coordinates>
  \n\t\t</Point>\n\t</Placemark>\n"
  #define WHEN  "\t\t\t<when>%s</when>\n"
  #define EARTH_RADIUS 6371
  using namespace std;
  vector<double> lons, lats;
  vector<char *> times;

  const char * trackName;
%}
```

```
%option noyywrap outfile="gps.cc"
%s IGNORE PATH
trkBegin \<trk\>.*
trkEnd \<\/trk\>.*
trkPtTime <time\>.*
trkPtBegin \<trkpt.*
trkPtEnd \<\/trkpt\>.*
trkPtSpeed \<speed\>.*
trkPtEle \<ele\>.*

%%
  double lat,lon,prevLat = 1000,prevLon = 1000,dlon,dlat;
  int pointNr = 0,noTime=1,speedValuesCount = 0;
  char time[30];
  float speedsSum = 0, speed, maxAltitude = -20000, minAltitude = 10000,altitude,distance = 0,a;
  void printStyles();

\<gpx.* {
          printf(HEADER);
          printStyles();
          BEGIN IGNORE;}
<IGNORE>{trkBegin} {
                    printf("\t<Placemark>\n\t\t<name>%s</name>\n\t\t<styleUrl>#line</styleUrl>\n\t\t
                    <gx:Track>\n",trackName);
                    BEGIN PATH;
                    }
<IGNORE>.|\n {}
<PATH>.|\n {}
<PATH>{trkPtBegin}  {
                  //https://andrew.hedges.name/experiments/haversine/ <- para calcular distância entre dois
                  pontos
                  sscanf(yytext,"<trkpt lat=\"%lf\" lon=\"%lf\">",&lat, &lon);
                  lons.push_back(lon);
                  lats.push_back(lat);
                  pointNr++;

                  //Conversão para radianos
                  lat = lat * M_PI / 180;
                  lon = lon * M_PI / 180;
                  if(prevLon != 1000 && prevLat != 1000){
                    dlon = lon - prevLon;
                    dlat = lat - prevLat;
                    a = pow(sin(dlat/2),2) + cos(lat) * cos(prevLat) * pow(sin(dlon/2),2);
                    distance += EARTH_RADIUS * 2 * atan2( sqrt(a), sqrt(1-a) ) ;
                  }

                  prevLon = lon;
                  prevLat = lat;
                }
<PATH>{trkPtTime} {
                  char * timeToAdd;
                  sscanf(yytext,"<time>%[^<]</time>",time);
                  printf(WHEN,time);
                  timeToAdd = strdup(time);
```

```
                  times.push_back(timeToAdd);
                  noTime = 0;
                  }
<PATH>{trkPtSpeed} {
                sscanf(yytext,"<speed>%f</speed>",&speed);
                speedsSum += speed;
                speedValuesCount++;
              }
<PATH>{trkPtEle} {
                sscanf(yytext,"<ele>%f</ele>",&altitude);
                if(altitude > maxAltitude)
                  maxAltitude = altitude;
                if(altitude < minAltitude)
                  minAltitude = altitude;
                }
<PATH>{trkPtEnd}  {
                  if(noTime)
                    fprintf( stderr, "One or more time elements missing");
                  else
                    noTime = 1;
                  }
<PATH>{trkEnd}  {
                  for(int i = 0; i < lats.size();i++){
                    lon = lons.at(i);
                    lat = lats.at(i);
                    printf(COORD,lon,lat,0.0);
                  }
                  printf("\t\t</gx:Track>\n");
                  printf("\t\t<description><![CDATA[");


                  printf("Distância(aprox): %f km<br>",distance);

                  if(times.size() > 0){
                    printf("Início: %s<br>",times.at(0));
                    printf("Fim: %s<br>",times.at(times.size()-1));
                  }

                  if(speedValuesCount > 0)
                    printf("Velocidade média: %.4f km/h<br>",speedsSum/(float)speedValuesCount);
                  else
                    printf("Velocidade média: N/A<br>");

                  if(maxAltitude != -20000 && minAltitude != 10000){
                    printf("Altitude máxima: %f m<br>",maxAltitude);
                    printf("Altitude mínima: %f m<br>",minAltitude);
                  }
                  else{
                    printf("Altitude máxima: N/A<br>");
                    printf("Altitude mínima: N/A<br>");
                  }
```

```
                printf("]]></description>\n");
                printf("\t</Placemark>\n");

                if(lons.size() > 0){
                  lon = lons.at(0);
                  lat = lats.at(0);
                  printf(MARKER,"Início",lat,lon,"#icon-green",lon,lat);

                  lon = lons.at(lons.size()-1);
                  lat = lats.at(lats.size()-1);
                  printf(MARKER,"Fim",lat,lon,"#icon-red",lon,lat);
                }
                printf("</Document>\n</kml>\n");
              }
%%
void printStyles(){
//Ícone verde normal

  printf("<Style id=\"icon-green-normal\">\n\t<IconStyle>\n\t\t<Icon>\n\t\t\t<href>http://www.gstatic.com/
  mapspro/images/stock/61-green-dot.png</href>\n\t\t</Icon>\n\t</IconStyle>\n</Style>\n");


//Ícone verde selecionado

  printf("<Style id=\"icon-green-highlight\">\n\t<IconStyle>\n\t\t<Icon>\n\t\t\t<href>http://www.gstatic.com
  /mapspro/images/stock/61-green-dot.png</href>\n\t\t</Icon>\n\t</IconStyle>\n\t<LabelStyle>\n\t\t<scale>1.1
  </scale>\n\t</LabelStyle>\n</Style>\n");


//Ícone verde

  printf("<StyleMap id=\"icon-green\">\n\t<Pair>\n\t\t<key>normal</key>\n\t\t<styleUrl>#icon-green-normal
  </styleUrl>\n\t</Pair>\n\t<Pair>\n\t\t<key>highlight</key>\n\t\t<styleUrl>#icon-green-highlight</styleUrl>
  \n\t</Pair>\n</StyleMap>\n");

//Ícone vermelho normal

  printf("<Style id=\"icon-red-normal\">\n\t<IconStyle>\n\t\t<Icon>\n\t\t\t<href>http://www.gstatic.com/
  mapspro/images/stock/123-red-dot.png</href>\n\t\t</Icon>\n\t</IconStyle>\n</Style>\n");

//Ícone vermelho selecionado

  printf("<Style id=\"icon-red-highlight\">\n\t<IconStyle>\n\t\t<Icon>\n\t\t\t<href>http://www.gstatic.com/
  mapspro/images/stock/123-red-dot.png</href>\n\t\t</Icon>\n\t</IconStyle>\n\t<LabelStyle>\n\t\t<scale>1.1
  </scale>\n\t</LabelStyle>\n</Style>\n");

//Ícone vermelho

  printf("<StyleMap id=\"icon-red\">\n\t<Pair>\n\t\t<key>normal</key>\n\t\t<styleUrl>#icon-red-normal
  </styleUrl>\n\t</Pair>\n\t<Pair>\n\t\t<key>highlight</key>\n\t\t<styleUrl>#icon-red-highlight</styleUrl>
  \n\t</Pair>\n</StyleMap>\n");

//Linha do trajeto
```

```
  printf("<Style id=\"line\">\n\t<LineStyle>\n\t\t<color>ffd18802</color>\n\t\t<width>5</width>\n\t
  </LineStyle>\n</Style>\n");

}

int main(int argc, const char* argv[]) {
  vector<double>::iterator itLons, itLats;
  int position;
  FILE *fr, *fw;
  if(argc>1){
    yyin=fopen(argv[1],"r");
    trackName = strndup(argv[1],strlen(argv[1])-4);
  }
  else
    trackName = "Trajeto";
  if(argc>2){
    fw =fopen(argv[2],"w");
    dup2(fileno(fw),1);
  }


  yylex();

  return 0;
}
```

Bibliografia

[1] Finding distances based on latitude and longitude. https://andrew.hedges.name/experiments/haversine/.

[2] Lexical analysis with flex, for flex 2.6.3. https://www.cs.virginia.edu/~cr4bd/flex-manual/.