Aprendizagem de Máquina

Resumo: Na fase 5 desenvolveremos programa que reconhece dígito manuscrito. Na fase 6, desenvolveremos um sistema que dirige carrinho autonomamente, de acordo com os dígitos que estiverem escritos nas placas. Para isso, utilizaremos o conceito de máquina de estados finita.

1 Introdução

A parte teórica de aprendizagem de máquina já foi visto na disciplina teórica. Veja as apostilas no site www.lps.usp.br/hae/apostila.

1. Aprendizagem de máquina (aprendizagem)
2. Aprendizagem de máquina avançada (mle_avancada)
3. Implementação de rede neural "do zero" (redeneural)
4. Rede neural convolutional (convolutional)

Em particular, o reconhecimento de dígitos manuscritos é abordado nas apostilas mle_avancada e convolutional.

2 Quinta fase do projeto

Modifique o programa fase3.cpp para que, além de detectar a placa, leia o dígito manuscrito inscrito dentro da placa quando a placa estiver suficientemente próximo da câmera. Fase5.cpp deve gravar o vídeo com a localização da placa e o dígito reconhecido.

Quadro de câmera capturado (capturado.avi).

Vídeo com placa localizada e dígito reconhecido (locarec.avi).

Figura 1: Localização da placa com reconhecimento do dígito manuscrito.

Note que, no interior da placa quadrado.png, há um quadrado menor amarelo pouco visível. Os dígitos manuscritos estão sempre dentro do quadrado amarelo. Durante a localização da placa, você deve considerar o conteúdo no interior do quadrado amarelo como "don't care". Só dentro do quadrado amarelo os pixels são (255, 255, 255). Os pixels brancos fora do quadrado amarelo são diferentes de (255, 255, 255).

Nota: Pode usar a base de dados de dígitos manuscritos MNIST para treinar o seu sistema de reconhecimento:

http://www.lps.usp.br/hae/apostilaraspi/mnist.zip

Site original: http://yann.lecun.com/exdb/mnist/

Nota: Dentro do Cekeikon, há a classe MNIST que faz a leitura da base de dados MNIST:

```
MNIST mnist(14, true, true);
// Reduz o tamanho das imagens para 14x14. inverte_cores=true, ajustaBbox=true
mnist.le("/home/hae/haebase/mnist");
// Faz a leitura do MNIST. O parametro eh o diretorio onde estao os arquivos.
```

Após a leitura, ficam disponíveis as seguintes estruturas:

```
int na; // Numero de amostras de treinamento
vector< Mat_<GRY> > AX; // Imagens de treinamento
                        // Ja redimensionadas, com cores invertidas, BBox.
vector<int> AY; // Classificacoes das imagens de treinamento (0 a 9)
Mat_<FLT> ax; // Imagens de treinamento convertidos para FLT, cada imagem numa linha.
Mat_<FLT> ay; // Classificacoes das imagens de treinamento convertidas para FLT (0 a 9)

int nq; // Numero de imagens testes
vector< Mat_<GRY> > QX; // Imagens de teste
                        // Ja redimensionadas, com cores invertidas, BBox.
vector<int> QY; // Classificacoes das imagens de teste (0 a 9)
Mat_<FLT> qx; // Imagens de teste convertidos para FLT, cada imagem numa linha.
Mat_<FLT> qy; // Classificacoes das imagens de teste  (0 a 9)
Mat_<FLT> qp; // Matriz ja alocado para colocar as classificacoes dos algoritmos de aprendizagem.
              // Matriz tem uma coluna e nq linhas.
```

Além disso, os seguintes métodos fazem redimensionamento de imagem (para tamanho especificado na construção da classe) e ajuste de bounding box. Não inverte as cores.

```
Mat_<GRY> MNIST::bbox(Mat_<GRY> a); // procura primeiro pixel diferente de 255
Mat_<FLT> MNIST::bbox(Mat_<FLT> a); // procura primeiro pixel menor que 0.5
```

Nota: Deixe todos os arquivos de entrada necessários no seu diretório default. Os arquivos necessários são:

`capturado.avi` (http://www.lps.usp.br/hae/apostilaraspi/capturado.avi)
`quadrado.png` (http://www.lps.usp.br/hae/apostilaraspi/quadrado.png)
`t10k-images.idx3-ubyte`
`t10k-labels.idx1-ubyte`
`train-images.idx3-ubyte`
`train-labels.idx1-ubyte`

Os quatro últimos arquivos são obtidos descompactando mnist.zip. Use obrigatoriamente os nomes dos arquivos acima. Executando:

```
$ fase5
```

o seu programa deve ler os arquivos necessários do diretório default e gerar o arquivo `locarec.avi` também no diretório default. Um exemplo de saída está em:

http://www.lps.usp.br/hae/apostilaraspi/locarec.avi

Use OpenMP para acelerar o processamento. Se quiser testar apenas o reconhecimento dos dígitos (sem a localização da placa), pode usar o vídeo abaixo com os dígitos já segmentados:

http://www.lps.usp.br/hae/apostilaraspi/digitos.avi

Item obrigatório (5 pontos): Mostre ao professor fase5 funcionando. Se não conseguir fazer este item durante a aula, deve trabalhar fora da aula e mostrar este item funcionando na aula seguinte.

3 Sexta fase do projeto

Nesta fase, pode ser que o seu programa (em desenvolvimento) termine sem que tenha desligado corretamente os motores. Para prevenir contra este imprevisto, é interessante deixar compilado um programa que desliga os motores.

```
//paramotor.cpp
//compila paramotor -w
#include <wiringPi.h>
int main () {
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  pinMode (1, OUTPUT) ;
  pinMode (2, OUTPUT) ;
  pinMode (3, OUTPUT) ;
  digitalWrite (0,  LOW) ;
  digitalWrite (1,  LOW) ;
  digitalWrite (2,  LOW) ;
  digitalWrite (3,  LOW) ;
}
```

Faça o sistema cliente-servidor cliente6.cpp e servidor6.cpp, onde o carrinho será controlado pelas placas afixadas em caixas. O carrinho deve seguir automaticamente o caminho indicado pelos dígitos manuscritos nas placas, com a convenção

2017:

0: Pare o carrinho.
1: Pare o carrinho.
2: Vire 180 graus à esquerda imediatamente.
3: Vire 180 graus à esquerda imediatamente.
4: Vire 180 graus à direita imediatamente.
5: Vire 180 graus à direita imediatamente.
6: Vire 90 graus à esquerda imediatamente.
7: Vire 90 graus à esquerda imediatamente.
8: Vire 90 graus à direita imediatamente.
9: Vire 90 graus à direita imediatamente.

2018:

0: Pare o carrinho.
1: Pare o carrinho.
2: Vire 180 graus à esquerda imediatamente.
3: Vire 180 graus à direita imediatamente.
4: Passe por baixo da placa e continue em frente.
5: Passe por baixo da placa e continue em frente.
6: Vire 90 graus à esquerda imediatamente.
7: Vire 90 graus à esquerda imediatamente.
8: Vire 90 graus à direita imediatamente.
9: Vire 90 graus à direita imediatamente.

O programa deve dirigir o carrinho autonomamente de acordo com as indicações das placas. Por exemplo:

```
raspberrypi$ servidor6
computador$ cliente6 192.168.0.110 [videosaida.avi]
```

Item obrigatório valendo nota: Mostre ao professor fase6 funcionando. Se não conseguir fazer este item durante a aula, deve trabalhar fora da aula e mostrar este item funcionando na aula seguinte.

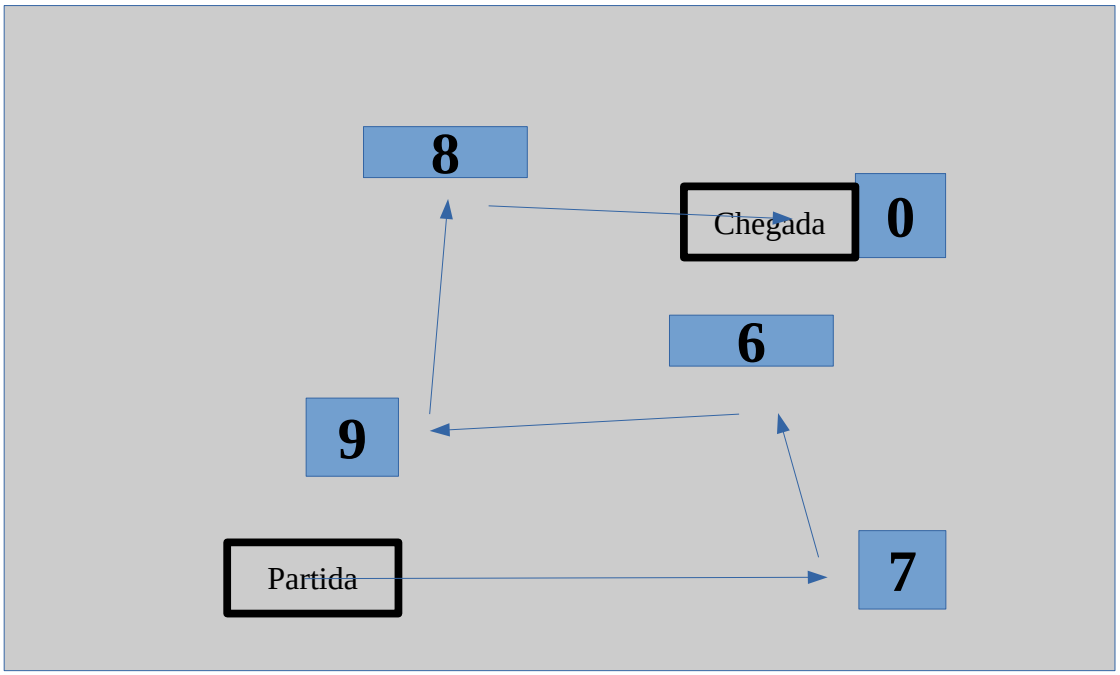

Figura 2: Demonstração 1 da fase 6.

Na demonstração da fase 6, o carrinho deve completar sozinho a trajetória das figuras 2, 3 e 4. Nas caixas, estarão afixadas placas com dígitos manuscritos. A trajetória do carrinho, entre uma placa e outra, será de aproximadamente 110 cm.

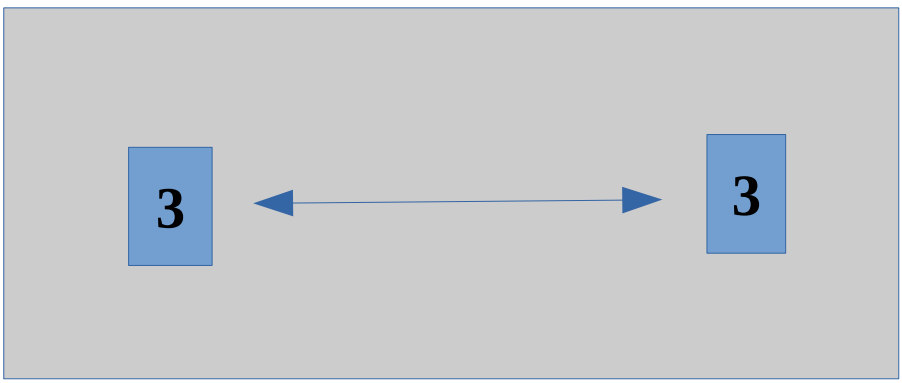

Figura 3: Demonstração 2 da fase 6: Carrinho vai e volta indefinidamente.

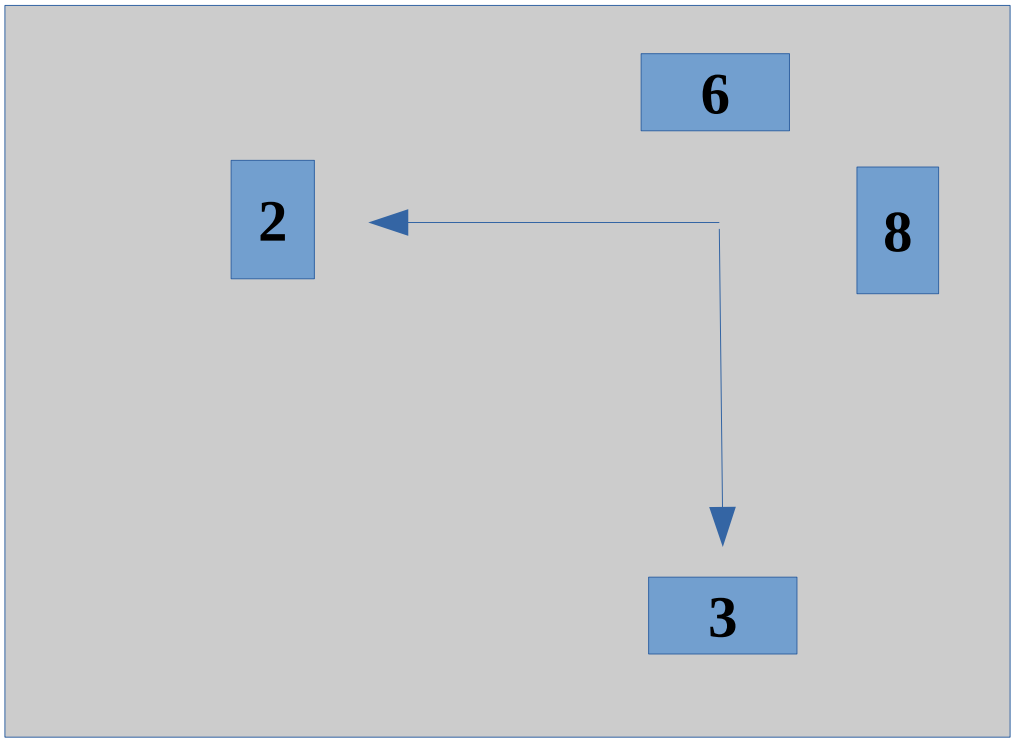

Figura 4: Demonstração da fase 6: Carrinho vai e volta indefinidamente fazendo curva.

Máquina de estados finita:

Uma ***máquina de estados finita*** (FSM - do inglês Finite State Machine) ou ***autômato finito*** é um modelo matemático usado para representar programas de computadores ou circuitos lógicos. É uma máquina abstrata que deve estar num estado de um conjunto finito de estados. A máquina pode estar num único estado por vez, e este estado é chamado de estado atual. Um estado armazena informações sobre o passado, isto é, ele reflete as mudanças desde o início do sistema, até o momento presente. Uma transição indica uma mudança de estado e é descrita por uma condição que precisa ser realizada para que a transição ocorra. Uma ação é a descrição de uma atividade que deve ser realizada num determinado momento. Máquinas de estado finitas podem modelar um grande número de problemas, entre os quais a automação de design eletrônico, projeto de protocolo de comunicação, análise e outras aplicações de engenharia [retirado de Wikipedia].

A fase 6 deste projeto pode ser modelada como uma máquina de estados. Se você tentar resolver esta fase usando programação convencional (estruturada ou orientada a objetos), sem modelá-lo como máquina de estados, provavelmente enfrentará muitas dificuldades.

A figura 5 mostra um exemplo de máquina de estados.

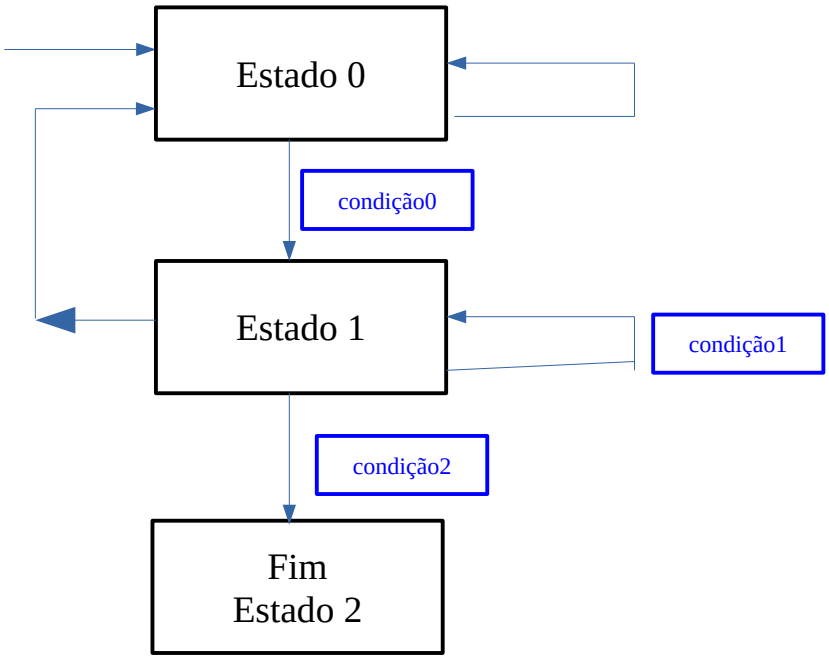

Figura 5: Um exemplo de máquina de estados.

A tradução do modelo de máquina de estados para a linguagem C++ pode usar comandos "goto":

```
estado0:
  //Tarefas para executar no estado0
  if (condicao0) goto estado1;
  else goto estado0;
estado1:
  //Tarefas para executar no estado1
  if (condicao1) goto estado0;
  else if (condicao2) goto estado2;
  else goto estado1;
estado2:
  //Termina o programa no estado2
```

Outra possibilidade é usar if/case dentro de um loop:

```
int estado=0;
do {
  if (estado==0) {
    //Tarefas para executar no estado0
    if (condicao0) estado=1;
  } else if (estado==1) {
    //Tarefas para executar no estado1
    if (condicao1) estado=0;
    else if (condicao2) estado=2;
    else estado=1;
  }
} while (estado!=2);
  //Termina o programa no estado2
```

O mais importante é modelar corretamente a máquina de estados, desenhando o diagrama de estados correto.

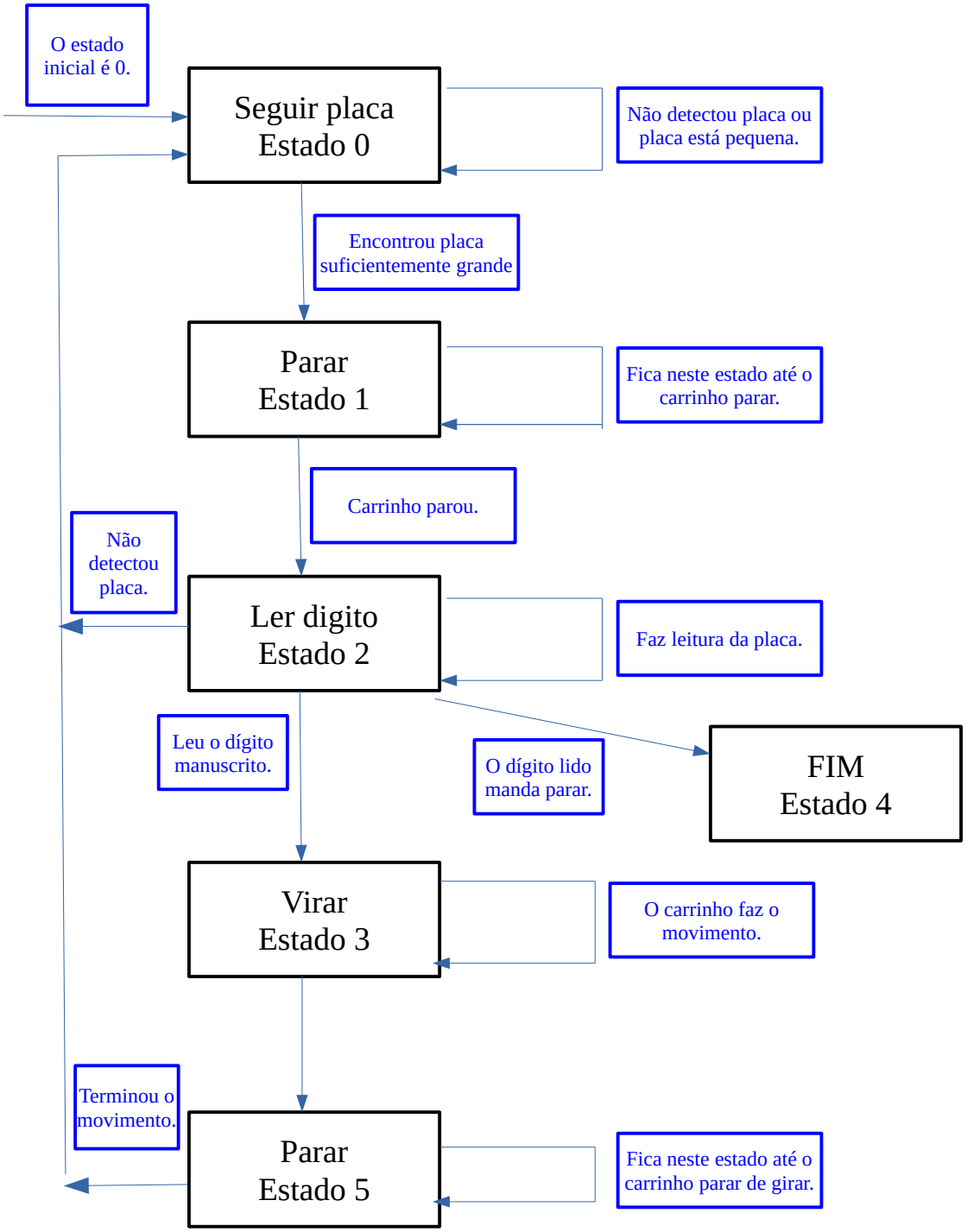

Figura 6: Um possível diagrama de estados (simplificado) da fase 6. Você deve escrever o seu próprio modelo.

Estado 0. No início do programa, o sistema está no estado 0. Este estado é semelhante ao projeto da fase 4. O sistema procura a placa. Se encontrar, o sistema faz o carrinho se movimentar em direção à placa. Se não encontrar, o sistema desliga os motores do carrinho. O sistema fica neste estado até que enxergue a placa suficientemente grande.

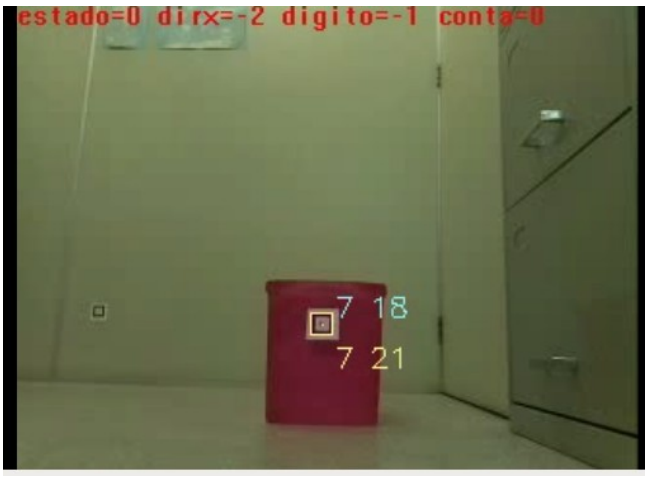

Figura 7: Estado 0 é o estado inicial do sistema. O carrinho procura placa e vai em sua direção.

Estado 1: O sistema entra neste estado quando detectar a placa suficientemente grande. Neste estado, o sistema manda parar os motores, para evitar uma colisão com a caixa. Repare que o carrinho não consegue parar instantaneamente. Assim, o sistema deve mandar parar o carrinho com certa antecedência. Na figura 8, o sistema tentou ler o dígito manuscrito, mas errou a leitura, pois a placa estava borrada pelo movimento do carrinho (o carrinho ainda não tinha freado completamente). O sistema fica neste estado o tempo suficiente até que o carrinho pare.

Figura 8: O sistema entra no estado 1 quando encontra placa suficientemente grande. O carrinho começa a parar.

Estado 2: O sistema entra no estado 2 um certo tempo após ter entrado no estado 1. Neste estado, o carrinho está completamente parado. A placa está bem visível com tamanho grande, sem borrão de movimento. Se não conseguir encontrar placa, o sistema deve voltar para o estado 0. Neste estado, faz a leitura do dígito e entra no estado 3 carregando a informação lida. Se o dígito lido indicar "pare", o programa termina.

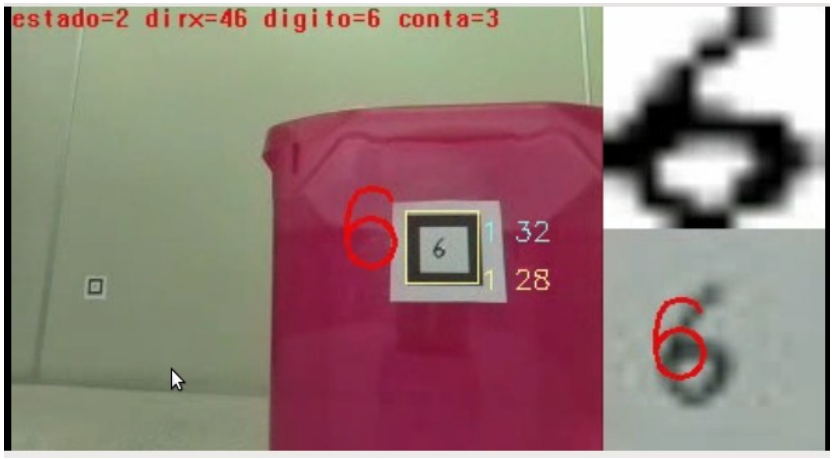

Figura 9: O sistema entra no estado 2 quando o carrinho parar completamente e enxergar a placa bem grande. Neste estado, faz a leitura do dígito manuscrito, com o carrinho completamente parado.

Estado 3: No estado 3, o carrinho faz o movimento indicado pelo dígito lido na fase 2 e depois volta para o estado 0.

Figura 10: No estado 3, o carrinho faz o movimento e volta para o estado 0.