Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

EA076M – Laboratório de Sistemas Embarcados

Prof. Dr. Fabiano Fruett

Projeto Final da Disciplina EA076M
Smart Classificador/Seletor de Grãos

Aluno: Gleyson Roberto do Nascimento. RA 043801.

Julho/2021

Apresentação de Projeto:

Considerando-se que o Brasil é um dos maiores produtores de grãos mundiais e que a cada ano a produtividade e as safras aumentam consideravelmente, a consolidação desta atividade como um dos pilares do PIB brasileiro é inegável.

Assim, considerando-se que na logística de colheita/transporte destes produtos há uma necessidade de classificação e seleção entre o que é grão e o que não é, para evitar custos adicionais com as impurezas e a desvalorização do produto final, este projeto é um smart classificador/seletor de grãos por machine learning.

1) Requisitos do cliente:

Embora não existam números oficiais sobre o quanto há de subproduto total numa safra, há uma classificação do Ministério da Agricultura quanto a quantidade de impurezas e sabe-se que este subproduto desvaloriza o preço final do produto. Contudo, boa parte das impurezas existe tentes são de origem orgânica e podem ser facilmente convertidas em adubo ou material combustível, sendo, portanto, rentáveis se bem selecionadas. Assim, do ponto de vista do cliente, é necessário o atendimento dos seguintes requisitos:

- Para que o valor final na venda dos grãos seja maximizado e o desperdício seja mínimo, deve haver um alto grau de certeza de classificação/seleção entre o que é e o que não é grão;
- O processo deve ser de fácil aplicação, manutenção e menor custo possível.

2) Requisitos do Projeto:

a) O que é?
Um sistema smart de classificação e seleção de grãos;

b) Para que serve?
O sistema busca a seleção e separação entre aquilo que é grão daquilo que não é grão, para a maximização do valor final de venda do produto, maximização do aproveitamento daquilo que é considerado impureza e minimização do desperdício;

c) Para quem é?
Para produtores de grãos em geral e também para prestadores de serviços logísticos que visam agregar valor ao produto transportado;

d) Onde será aplicado?
Este sistema pode ser aplicado tanto no momento da colheita de grãos, embarcado em colheitadeiras, quanto no sistema de transporte, embarcado em esteiras de transporte;

e) Quando será aplicado?
Neste caso é um projeto acadêmico, sem maiores pretensões de ser diretamente aplicado, mas para fins didáticos, a apresentação ocorre em julho/2021;

f) Como será realizado?
De forma simplificada, a classificação será por machine learning com base numa entrada de imagem de webcam e a seleção de caminho por servomotor.

g) De que forma será realizado?

O projeto seguirá o seguinte esquemático:

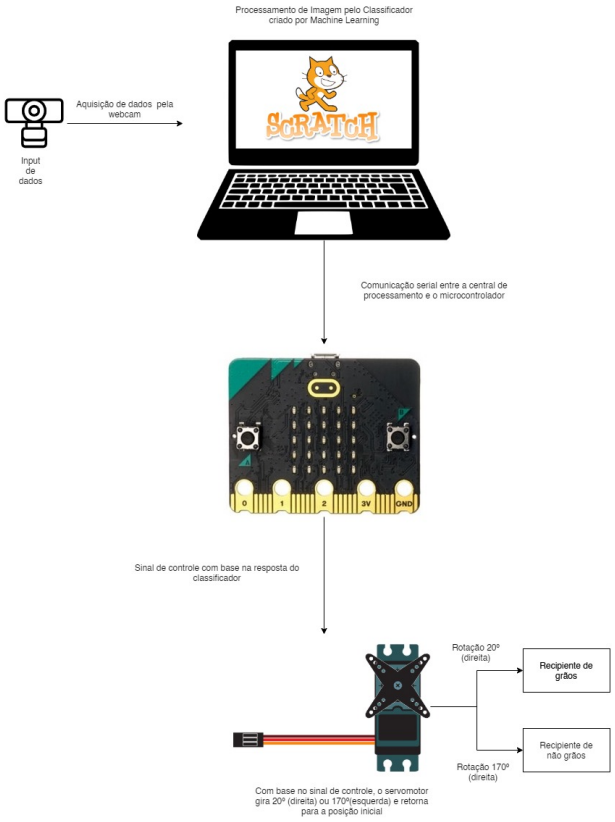

Figura 01 – Esquemático do Projeto.

3) Diagrama Funcional do Projeto:

O projeto seguirá o seguinte diagrama funcional:

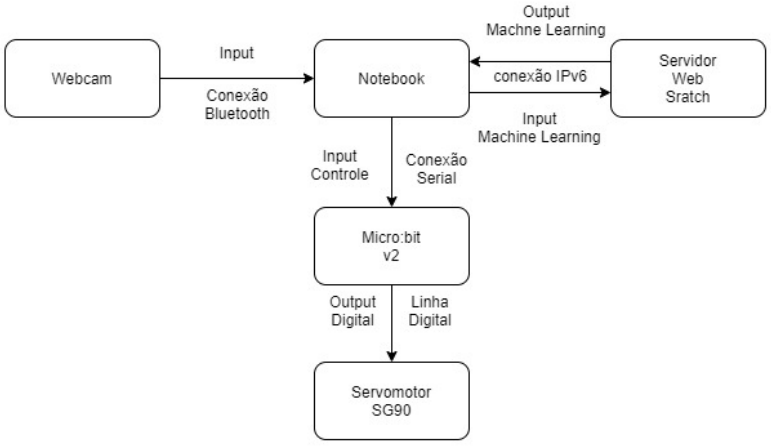

Figura 02 – Diagrama Funcional do Projeto.

4) Diagrama de Hardware:

O projeto seguirá o seguinte diagrama de hardware:

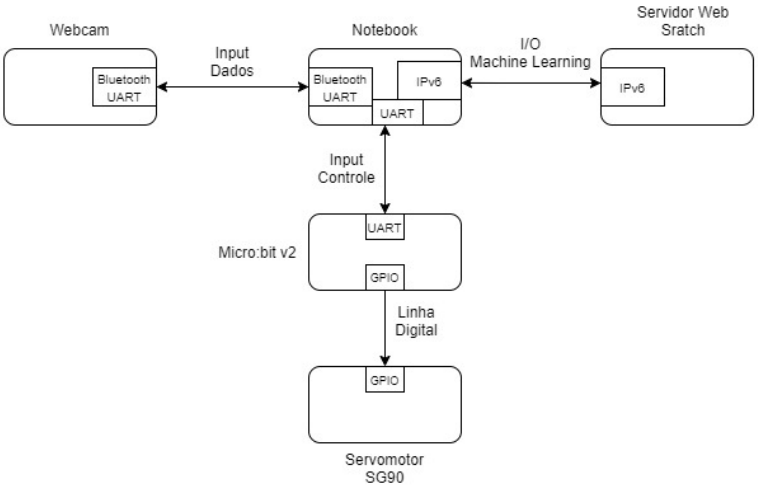

Figura 03 – Diagrama de Hardware do Projeto.

5) Lista de Materiais – BOM:

Material	Qtd	Unidade
Notebook com webcam e acesso a internet para aquisição, processamento e classificação dos dados de entrada	1	un
Protoboard	1	un
Adaptador para conexão entre Microbit com o Protoboard	1	un
Servomotor SG90 para a seleção da amostra em grão e não grão	1	un
Jumper macho-macho	5	un
Jumper macho-fêmea	5	un
Jumper fêmea-fêmea	5	un

6) Aquisição e orçamento dos componentes:

Excetuando-se o notebook e a webcam que são de propriedade do aluno, os demais materiais que constam na Lista de Materiais – BOM foram gentilmente emprestados pelo Prof. Fabiano Fruett, responsável pela disciplina, de forma que os mesmos já estão sob a guarda do aluno até a apresentação do projeto para posterior devolução. Assim, o custo final do projeto será zero.

7) Diagrama de Firmware/Software:

O projeto seguirá o seguinte fluxograma:

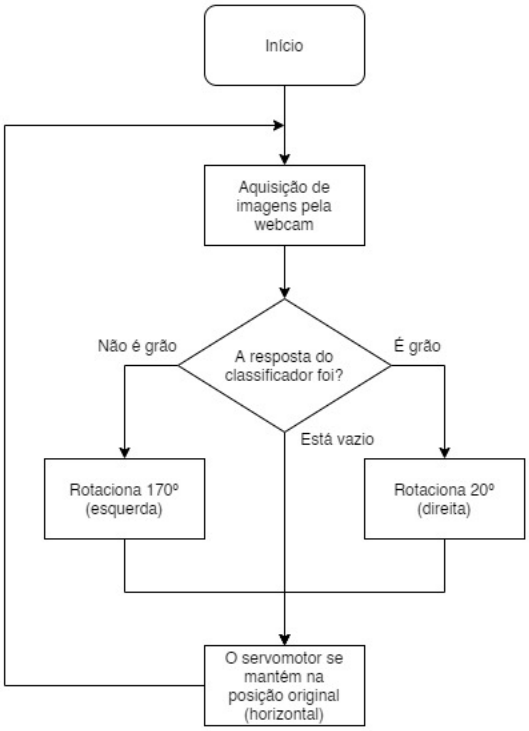

Figura 04 – Fluxograma do Projeto.

Softwares utilizados:

Para a criação do modelo de classificador por machine learning, foi utilizado o Python 3.7, através da biblioteca Tensorflow e o modelo resultante foi agregado na nuvem do Google no endereço https://teachablemachine.withgoogle.com/models/-URltWzWh/

Para programação geral do Microbit e manipulação do Tensorflow Lite será utilizado o sistema web do Scratch na versão 3.

Desenvolvimento do Protótipo:

1.1- Rede Neural Convolucional (CNN):

Nesta primeira etapa do protótipo, o enfoque foi na elaboração de um bom classificador por *deep learning*, assim, como os dados de input seriam imagens, seria mais adequado para a classificação o uso de uma rede neural convolucional (CNN). Através do Python, pela biblioteca Tensorflow, foi realizada a CNN com uma arquitetura similar ao modelo Imagenet, existente no Tensorflow:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 85, 85]           9,408
```

```
       BatchNorm2d-2           [-1, 64, 85, 85]             128
              ReLU-3           [-1, 64, 85, 85]               0
         MaxPool2d-4           [-1, 64, 43, 43]               0
            Conv2d-5           [-1, 64, 43, 43]          36,864
       BatchNorm2d-6           [-1, 64, 43, 43]             128
              ReLU-7           [-1, 64, 43, 43]               0
            Conv2d-8           [-1, 64, 43, 43]          36,864
       BatchNorm2d-9           [-1, 64, 43, 43]             128
             ReLU-10           [-1, 64, 43, 43]               0
       BasicBlock-11           [-1, 64, 43, 43]               0
           Conv2d-12           [-1, 64, 43, 43]          36,864
      BatchNorm2d-13           [-1, 64, 43, 43]             128
             ReLU-14           [-1, 64, 43, 43]               0
           Conv2d-15           [-1, 64, 43, 43]          36,864
      BatchNorm2d-16           [-1, 64, 43, 43]             128
             ReLU-17           [-1, 64, 43, 43]               0
       BasicBlock-18           [-1, 64, 43, 43]               0
           Conv2d-19          [-1, 128, 22, 22]          73,728
      BatchNorm2d-20          [-1, 128, 22, 22]             256
             ReLU-21          [-1, 128, 22, 22]               0
           Conv2d-22          [-1, 128, 22, 22]         147,456
      BatchNorm2d-23          [-1, 128, 22, 22]             256
           Conv2d-24          [-1, 128, 22, 22]           8,192
      BatchNorm2d-25          [-1, 128, 22, 22]             256
             ReLU-26          [-1, 128, 22, 22]               0
       BasicBlock-27          [-1, 128, 22, 22]               0
           Conv2d-28          [-1, 128, 22, 22]         147,456
      BatchNorm2d-29          [-1, 128, 22, 22]             256
             ReLU-30          [-1, 128, 22, 22]               0
           Conv2d-31          [-1, 128, 22, 22]         147,456
      BatchNorm2d-32          [-1, 128, 22, 22]             256
             ReLU-33          [-1, 128, 22, 22]               0
       BasicBlock-34          [-1, 128, 22, 22]               0
           Conv2d-35          [-1, 256, 11, 11]         294,912
      BatchNorm2d-36          [-1, 256, 11, 11]             512
             ReLU-37          [-1, 256, 11, 11]               0
           Conv2d-38          [-1, 256, 11, 11]         589,824
      BatchNorm2d-39          [-1, 256, 11, 11]             512
           Conv2d-40          [-1, 256, 11, 11]          32,768
      BatchNorm2d-41          [-1, 256, 11, 11]             512
             ReLU-42          [-1, 256, 11, 11]               0
       BasicBlock-43          [-1, 256, 11, 11]               0
           Conv2d-44          [-1, 256, 11, 11]         589,824
      BatchNorm2d-45          [-1, 256, 11, 11]             512
             ReLU-46          [-1, 256, 11, 11]               0
           Conv2d-47          [-1, 256, 11, 11]         589,824
      BatchNorm2d-48          [-1, 256, 11, 11]             512
             ReLU-49          [-1, 256, 11, 11]               0
       BasicBlock-50          [-1, 256, 11, 11]               0
           Conv2d-51            [-1, 512, 6, 6]       1,179,648
      BatchNorm2d-52            [-1, 512, 6, 6]           1,024
             ReLU-53            [-1, 512, 6, 6]               0
           Conv2d-54            [-1, 512, 6, 6]       2,359,296
      BatchNorm2d-55            [-1, 512, 6, 6]           1,024
           Conv2d-56            [-1, 512, 6, 6]         131,072
      BatchNorm2d-57            [-1, 512, 6, 6]           1,024
             ReLU-58            [-1, 512, 6, 6]               0
       BasicBlock-59            [-1, 512, 6, 6]               0
           Conv2d-60            [-1, 512, 6, 6]       2,359,296
      BatchNorm2d-61            [-1, 512, 6, 6]           1,024
             ReLU-62            [-1, 512, 6, 6]               0
           Conv2d-63            [-1, 512, 6, 6]       2,359,296
      BatchNorm2d-64            [-1, 512, 6, 6]           1,024
             ReLU-65            [-1, 512, 6, 6]               0
       BasicBlock-66            [-1, 512, 6, 6]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                  [-1, 2]           1,026
================================================================
Total params: 11,177,538
Trainable params: 11,177,538
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.33
```

```
Forward/backward pass size (MB): 37.72
Params size (MB): 42.64
Estimated Total Size (MB): 80.69
----------------------------------------------------------------
```

Figura 05 – Arquitetura da CNN.

Todavia, para as imagens de milho/café e vazio, a acurácia média encontrada para esta rede foi de 0.78, assim, o optou-se pelo modelo keras_model.h5 existente na rede pré-treinada do Google, que em termos de arquitetura é bastante semelhante ao Imagenet, todavia, o dataset base do Google é muito superior em termos de diversidade e atualização de forma que houve um avanço considerável de acurácia, assim, foram obtidos os seguintes resultados:

Accuracy per class

CLASS	ACCURACY	# SAMPLES
vazio	1.00	5
cafe	1.00	7
milho	1.00	7

Figura 06 – Tabela de acurácia da CNN.

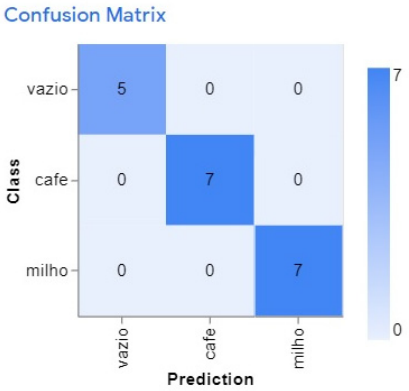

Figura 07 – Matriz de Confusão da CNN.

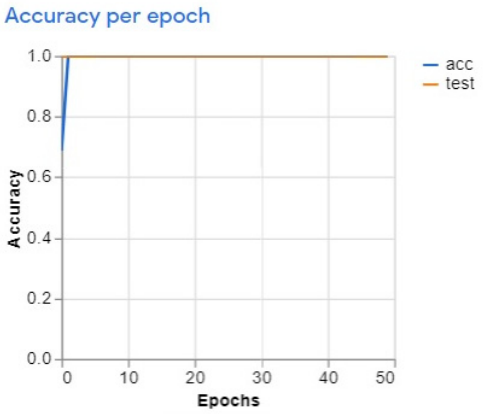

Figura 08 – Acurácia por época na CNN.

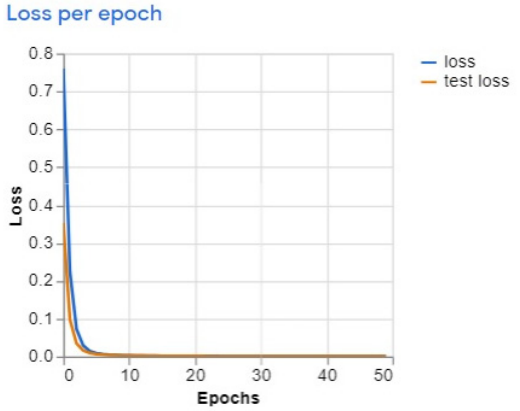

Figura 09 – Loss por época na CNN.

Desta forma, embora a tendência de pensamento fosse o *overfitting*, considerando-se que o dataset base do Google e o modelo possuem atualização constante e cada vez mais imagens, a especialização é bastante possível para imagens mais simples, como é o caso do grão de milho, café e também o vazio.

A programação em Python realizada foi:

```
import tensorflow.keras
from PIL import Image, ImageOps
import numpy as np

# retirando a notificação
```

```
np.set_printoptions(suppress=True)

# carregando o modelo para transfer leraning
model = tensorflow.keras.models.load_model('keras_model.h5')

#criando o dataset
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

# Abrindo as imagens salvas pela webcam
image = Image.open('webcam.jpg')

#fazendo resize e antialias da imagem
size = (224, 224)
image = ImageOps.fit(image, size, Image.ANTIALIAS)

#convertendo a imagem em array numérico
image_array = np.asarray(image)

# mostrando a imagem
image.show()

# normalizando a imagem
normalized_image_array = (image_array.astype(np.float32) / 127.0) - 1
# carregando a imagem normalizada
data[0] = normalized_image_array

# fazendo a predição
prediction = model.predict(data)
print(prediction)
```

Desta forma, após a finalização do modelo ele foi upado na nuvem do Google no endereço:
https://teachablemachine.withgoogle.com/models/-URltWzWh/

1.2- Programação no Scratch:

No Scratch, era necessário o controle da webcam para que houve a classificação, assim, através do módulo de AI, foi possível trazer o modelo do classificador pelo seu endereço na nuvem do Google e assim ter acesso as classes de classificação. Uma vez com estes dados, para cada classe foi controlado um ângulo de servomotor no Microbit e uma mensagem no display do microcontrolador indicando a classificação.

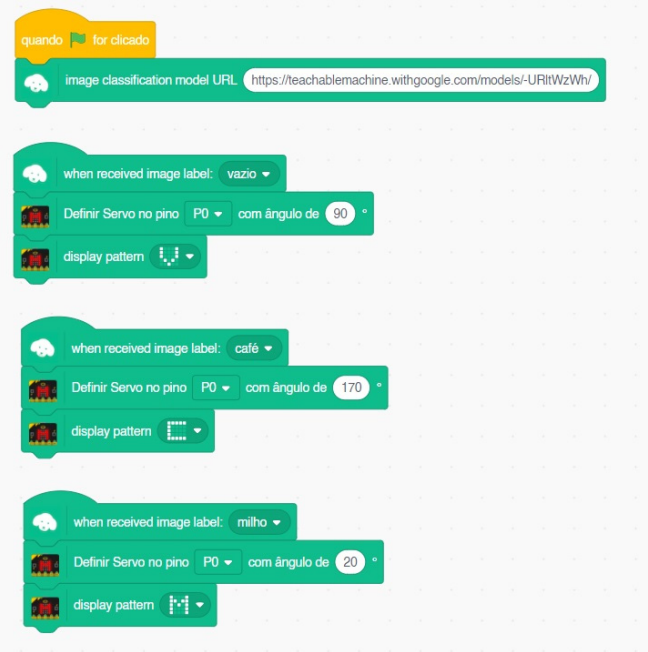

Figura 10 – Programação no Scratch.

1.3- Hardware - Protótipo de validação:

Neste primeiro momento, foi realizada uma montagem mais simples do Microbit e servomotor, apenas para a validação do modelo, assim, o microcontrolador foi acoplado ao adaptador para Protoboard e foram conectados jumpers diretamente para comunicação com o servomotor, como segue:

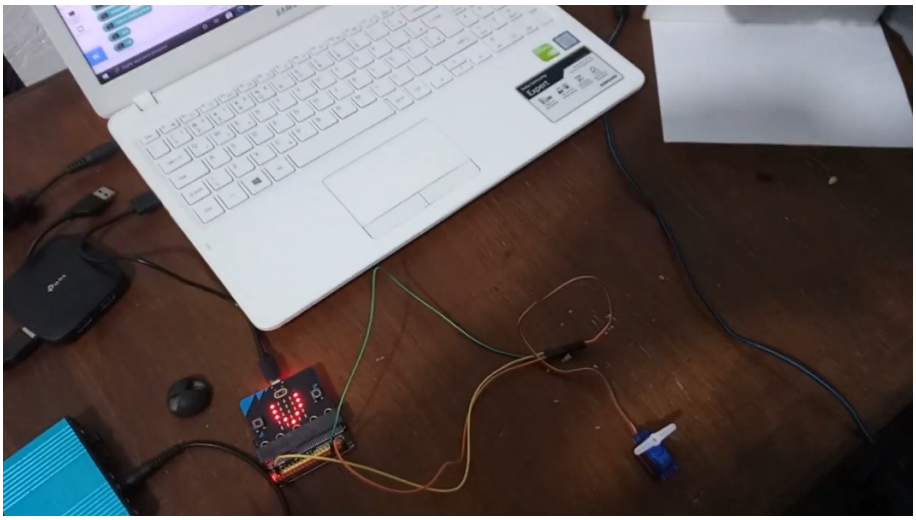

Figura 11 – Montagem simplificada do Microbit e servomotor.

Todo o processo realizado nesta fase inicial de prototipagem encontra-se em vídeo no Youtube: https://www.youtube.com/watch?v=hHSuHq7_Gpg

1.4- Hardware – Montagem Final:

O protótipo final foi montado conforme o esquemático abaixo:

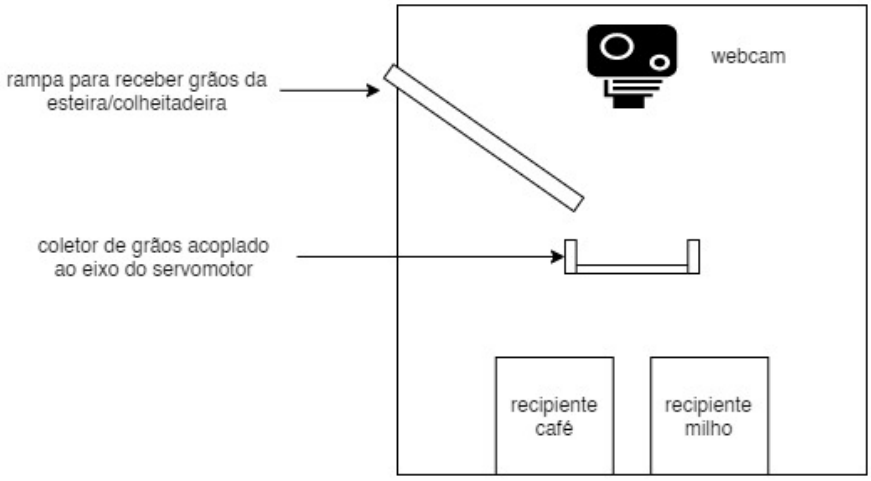

Figura 12 – Esquemático da montagem do protótipo final.

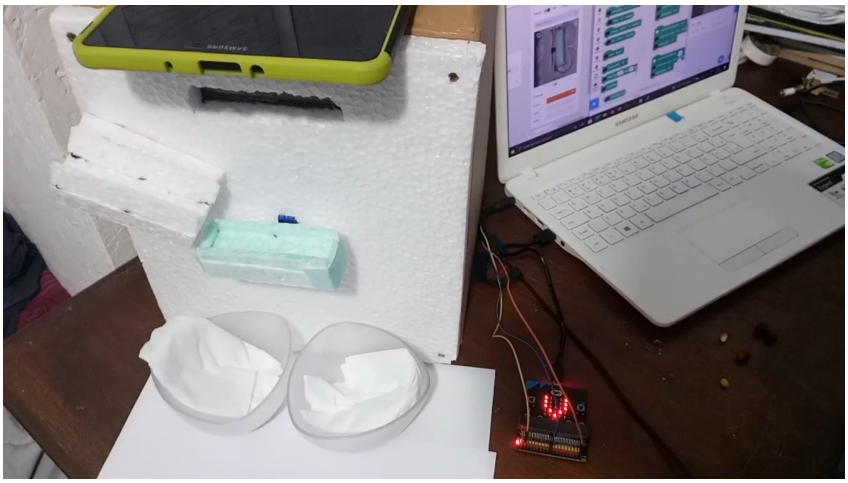

Figura 13 – Montagem final do protótipo proposto.

Todo o processo realizado nesta fase final de prototipagem encontra-se em vídeo no Youtube: https://www.youtube.com/watch?v=pW3JfBz177k e encontra-se no repositório do Github: https://github.com/grnbatera/EA076

Avanços e dificuldades encontrados:

Podemos citar como dificuldades e avanços os seguintes pontos no projeto:

- *Encontrar uma rede pré-treinada e um bom dataset para que as acurácias das imagens fossem positivas:* neste primeiro momento esta foi a dificuldade maior e fez com que alguns sistemas, como foi o caso do Edge Impulse, fossem descartados; todavia, a rede imagenet trouxe uma acurácia melhor e consequentemente levou

até a rede pré-treinada e o dataset do Google, assim, através do processo de transfer learning, o processo se tornou facilitado e de maior acurácia;

- *Overfitting:* Esse sem dúvida foi um grande problema que a princípio parecia ocorrer no projeto, todavia, após uma análise melhor e mais testes, chegou-se à conclusão que a rede pré-treinada e o dataset do Google eram muito poderosos para imagens mais simples como foi o caso das classes existentes parta este projeto, então, como nos testes preliminares não houve erro com diferentes tipos de grãos de milho e café, supôs-se que de fato a acurácia de 100% fosse possível, sendo uma quebra de paradigma, já que normalmente a acurácia 100% é sinal de *overfitting* em condições e redes mais comuns;

- *Comunicação com o Microbit e o Scratch:* num primeiro momento parecia que esta comunicação seria bastante tranquila, contudo, a versão v2 do Microbit precisa de um arquivo .hex (encontrado em https://microbit-more.github.io/) específico para aceitar a comunicação com o Scratch e habilitar os módulos disponíveis no Scratch que expandem as possibilidades de trabalho do Microbit. Desta forma, embora tenha sido bastante trabalhoso descobrir qual era o erro na comunicação e a necessidade do arquivo .hex específico, uma vez solucionado, fez do Microbit uma ferramenta com mais possibilidades do que no uso do Thinkercad e Microbit.org.