Introdução a Robótica
Relatório do Trabalho Prático 1

Grupo 07 – OMC.

Componentes: Henrique Silva Chaltein de Almeida
Mauro Antônio da Costa Júnior
Otávio Cardoso Borges

Professor: Mário Fernando Montenegro Campos

1. Proposta

Sendo este o primeiro trabalho, seu objetivo principal foi apresentar aos alunos as ferramentas básicas sobre as quais os demais trabalhos serão desenvolvidos. As atividades propostas incluem montar um robô baseado na "handybug" (montagem especificada no livro Robotics Exploration, de Fred Martin) acrescentando uma caneta para permitir que o robô desenhe algumas trajetórias pré-estabelecidas. Visando manter a tarefa em nível introdutório, as únicas formas exigidas foram linhas retas, círculos e quadrados. A avaliação consiste em programar o robô para desenhar uma seqüência de três quadrados e outra de três círculos.

2. Elaboração de um plano de trabalho.

As primeiras discussões do grupo buscaram a elaboração de uma estratégia de de projeto. Duas idéias principais prevaleceram nesta etapa:

- A caneta deveria ficar sobre o eixo de tração e em uma posição simétrica às rodas
- A velocidade de cada roda deveria ser monitorada por algum sensor.

A primeira proposta visava facilitar os cálculos de posicionamento da caneta, a simetria garante que o movimento desejado do ponto de contato com o papel possa ser facilmente mapeado como um movimento o robô. Para tanto, a caneta deveria ser colocada no centro da montagem, porém houveram complicações neste aspecto. Devido às dimensões consideráveis da handyboard em relação às demais partes da estrutura, ela ocuparia a maior parte da área superior do robô. Assim, para colocar a caneta no centro, esta deveria ficar completamente abaixo da handyboard, e se desejássemos mantê-la em posição vertical, teríamos que aumentar bastante a altura do robô. Tal alteração na estrutura não nos pareceu viável por não haverem peças verticalmente longas no kit, forçando ao uso de diversas partes emendadas, o que poderia comprometer a estabilidade da estrutura mecânica da montagem. Uma possível solução proposta foi colocar a caneta inclinada, porém logo abandonamos esta idéia pelo fato do espaço central logo abaixo da handyboard abrigar os motores. Se a caneta fosse colocada ali diagonalmente, seria difícil fazer uma estrutura mecânica rígida em espaço tão reduzido, e tornaria inviável a proposta de adicionar um grau de liberdade à caneta, para que ela riscasse apenas quando fosse necessário.

Decidimos então manter a posição de escrita da caneta simétrica em relação às rodas, porém transladada em relação ao eixo de tração. A montagem correspondente a esta proposta será ilustrada na próxima seção.

O segundo conceito de montagem visa aumentar a robustez do robô a imperfeições do modelo, como as diferenças entre os motores, a variação de carga na bateria, diferenças de aderência entre as rodas e variações de parâmetros. Foi discutido em sala o uso de calibração e estimativas de temporização, o que caracteriza um funcionamento em malha aberta. No entanto, não foi especificado um modelo de montagem a seguir, assim, mesmo o uso de

sensores para realimentação não sendo tema deste trabalho, seu uso foi considerado válido. Além da melhoria em desempenho, outra vantagem desta técnica é reduzir bastante o tempo gasto nas calibrações do robô, pois o sistema é capaz de corrigir em tempo real erros causados pela diferença entre parâmetros modelados e reais.

3. Montagem

Inicialmente, seguimos a montagem descrita no capítulo dois do livro-texto, a partir da página 29. Uma das primeiras diferenças surgiu na montagem dos motores. O kit fornecido na disciplina traz apenas um motor lego, então seria necessário adaptar um motor comum para completar a montagem. Este seria montado sobre uma placa lego cortada de modo a nivelar a altura de ambos. Decidimos por adaptar dois motores à montagem e não usar o do kit, pois seria mais trabalhoso fazer o cabeamento adequado a ele, além das restrições ao uso de cola quente e outros elementos que pudessem danificar o kit. Na figura 1 podemos ver os motores posicionados na montagem.

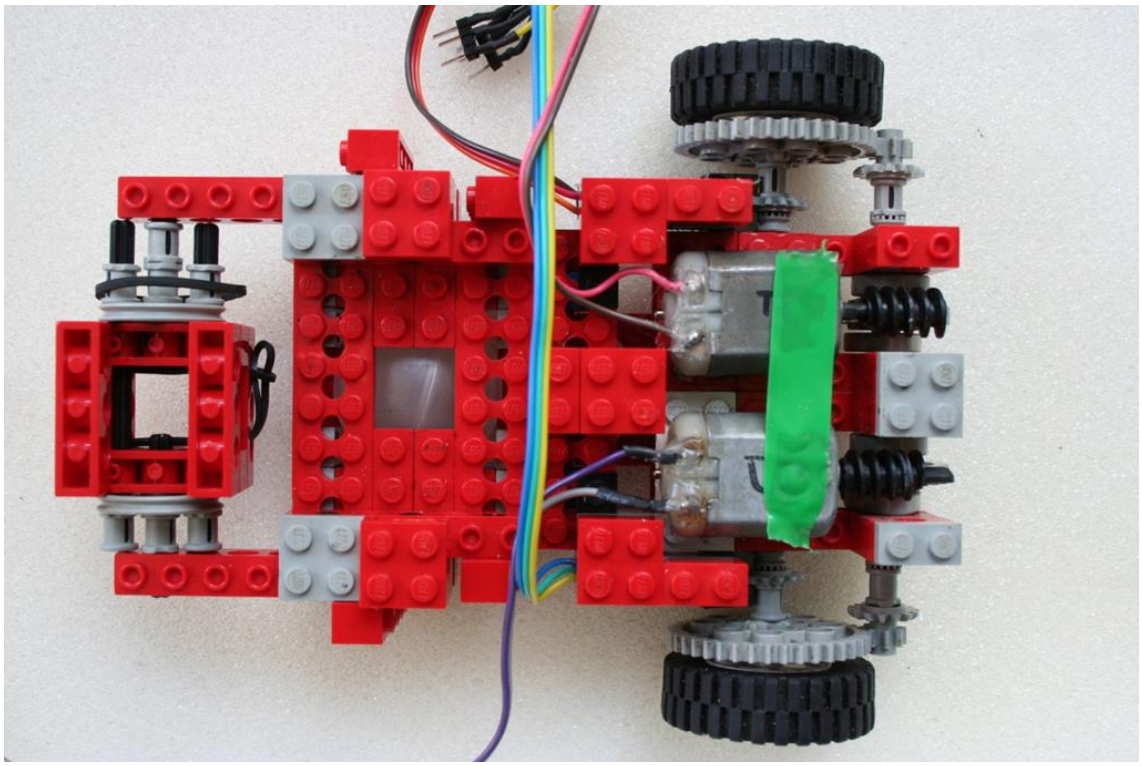

Figura 1: Visão geral da estrutura interna do robô

Uma fita adesiva dupla face foi colada sobre os dois motores para melhorar sua sustentação e garantir que a placa a ser montada sobre esta parte os pressionasse de modo a manter esta posição.

Outra diferença fundamental em relação à handybug foi a substituição das duas rodas de sustentação por um roll-on. Esta característica foi sugerida pelo professor e usada por todos os grupos, pois confere ao robô uma reposta melhor em curvas do que rodas paralelas enquanto mantém uma boa insensibilidade a imperfeições da superfície de apoio. No entanto, deve-se observar se a esfera do roll-on não está presa, pois o desempenho da montagem fica bastante comprometido se ela desliza ao invés de girar. Um destaque desta etapa foi conseguir fixar o roll-on na estrutura sem a necessidade de colá-lo em nenhuma peça lego. Ele simplesmente foi encaixado em uma estrutura que prendia sua base, limitando a movimentação perpendicular ao plano da base. A traseira desta montagem foi coberta com

algumas peças para evitar recuos. Tomamos ainda o cuidado de não pressioná-lo lateralmente para evitar atrito da esfera com seu apoio. As figuras 2 e 3 mostram o roll-on em seu suporte.

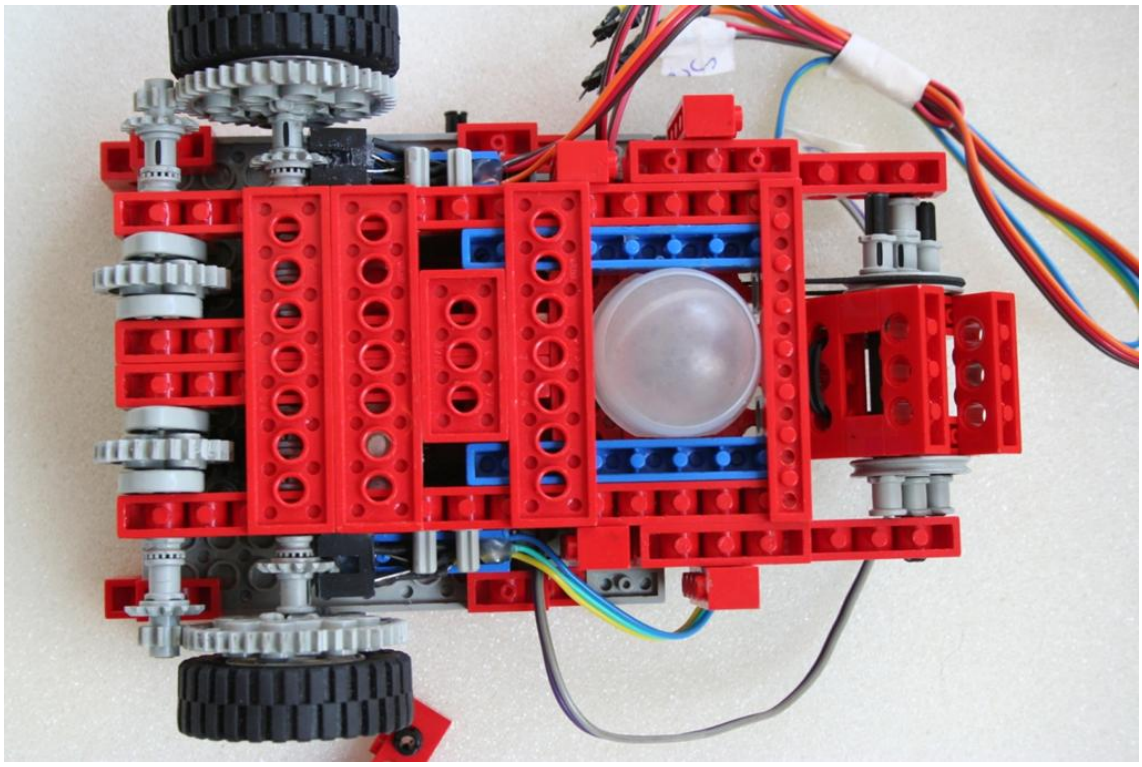

Figura 2: vista da parte de baixo do robô

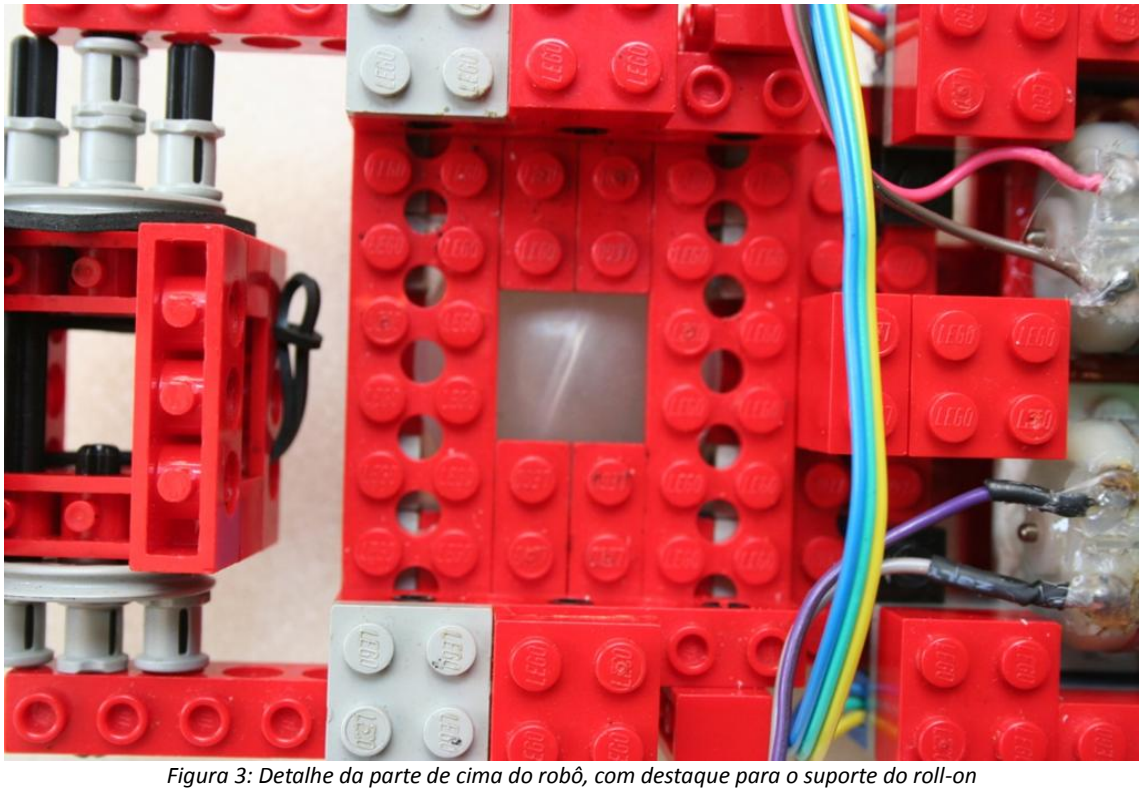

Figura 3: Detalhe da parte de cima do robô, com destaque para o suporte do roll-on

A handybug incluía também uma espécie de pára-choques, que acionava dois switchs, indicando o contato com objetos. Em nossa montagem esta peça foi removida, e foi montado um suporte para a caneta em seu lugar. Este pode ser visto em todas as figuras apresentadas até agora, sendo a peça semelhante a um cubo cortado por eixos, montada na frente do robô. A caneta é encaixada no furo central da peça, e presa com um elástico (mostrado nas figuras). Os eixos colocados nesta caixa a permitem girar livremente para qualquer lado, e a tração da peça é feita por um terceiro motor, ligado a esta por um elástico preso a uma das polias deste suporte. Este motor pode ser visto na figura 4.

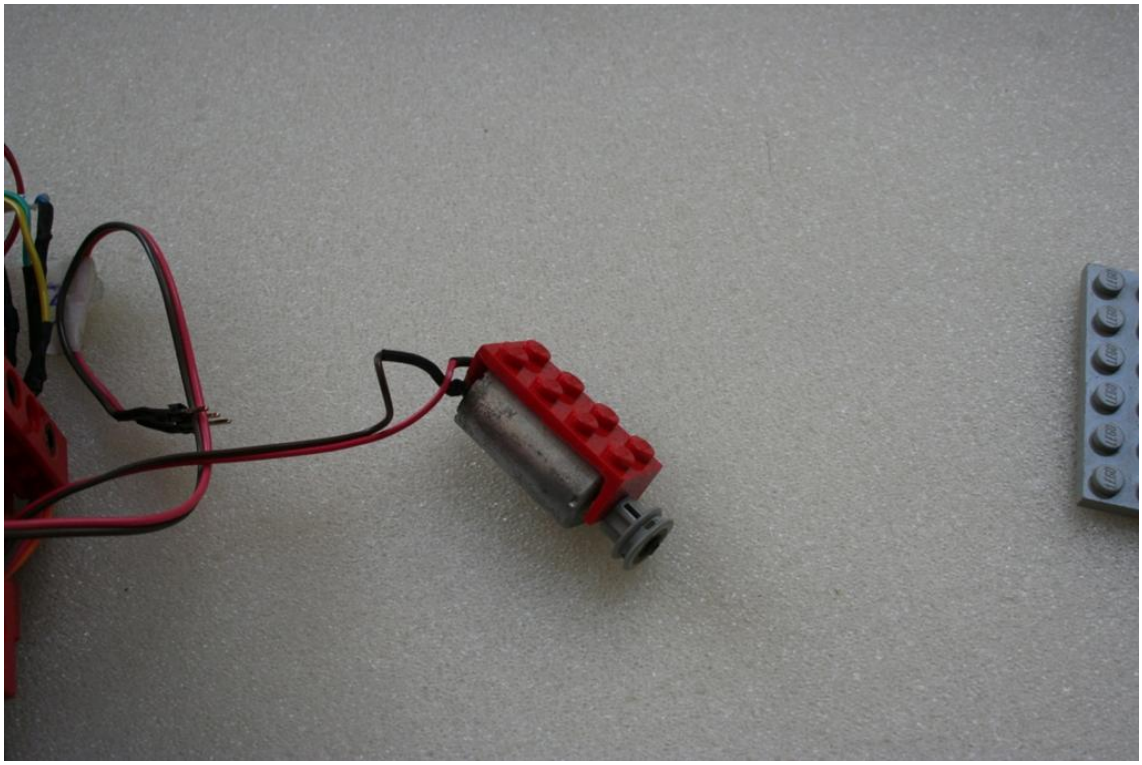

Figura 4: Motor responsável por mover a caneta

Devido a restrições no alinhamento e comprimento do elástico que liga as polias, o motor adicional foi montado sobre o suporte do roll-on, mas sua fixação usou a peça localizada acima dele. A figura 5 ilustra esta configuração.

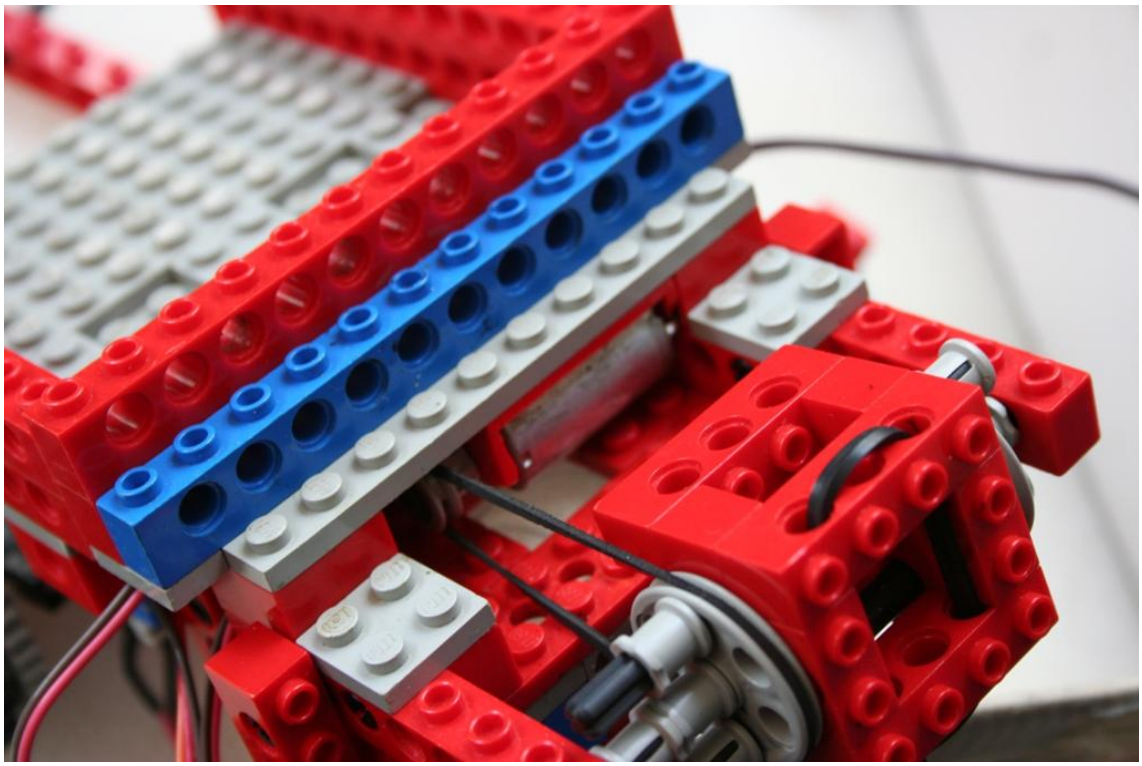

Figura 5: sistema de posicionamento da caneta

Para medir a velocidade do robô, alongamos o eixo das rodas e adicionamos uma engrenagem do tipo bevel e um sensor de interrupção de feixe a cada eixo, conforme pode ser visto nas figuras 1 e 2, e evidenciado no detalhe a seguir.

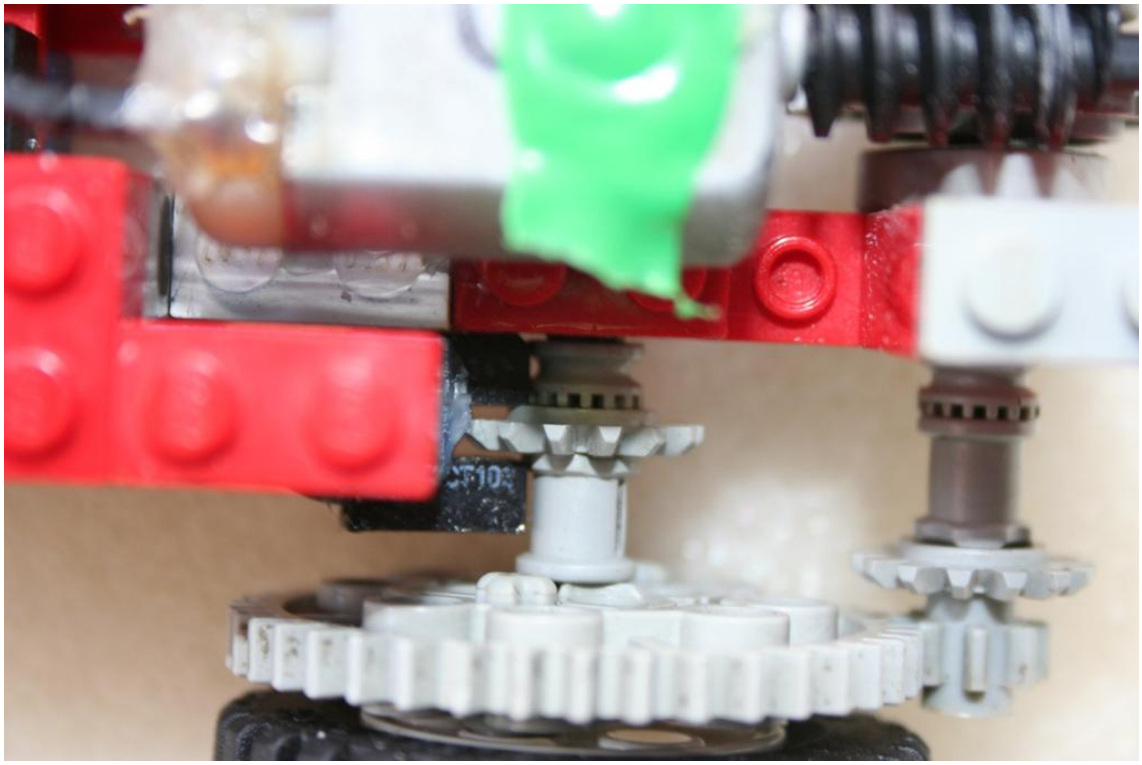

Figura 6: Sensor de interrupção de feixe para a medição de deslocamento angular o eixo.

Além do encaixe na parte inferior, os o braço de apoio dos sensores também é fixado pela placa que cobre a região dos motores. As figuras 7 e 8 ilustram esta característica.

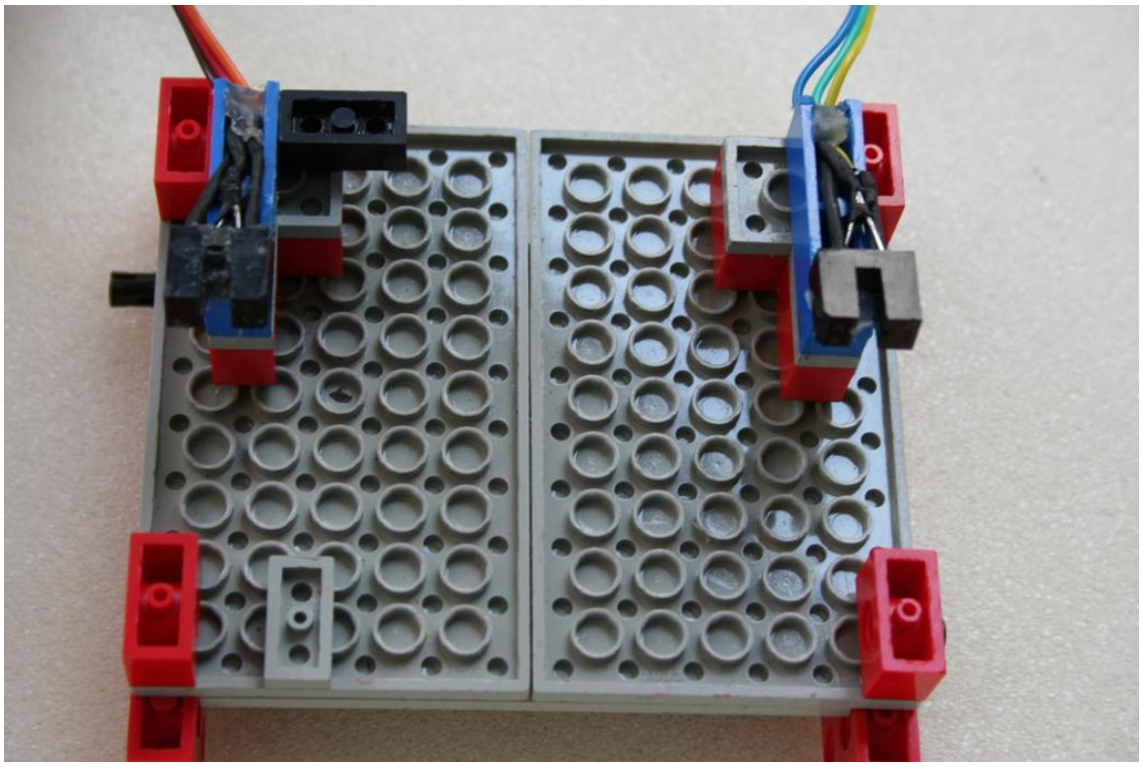

Figura 7: Placa de apoio da handyboard vista por baixo

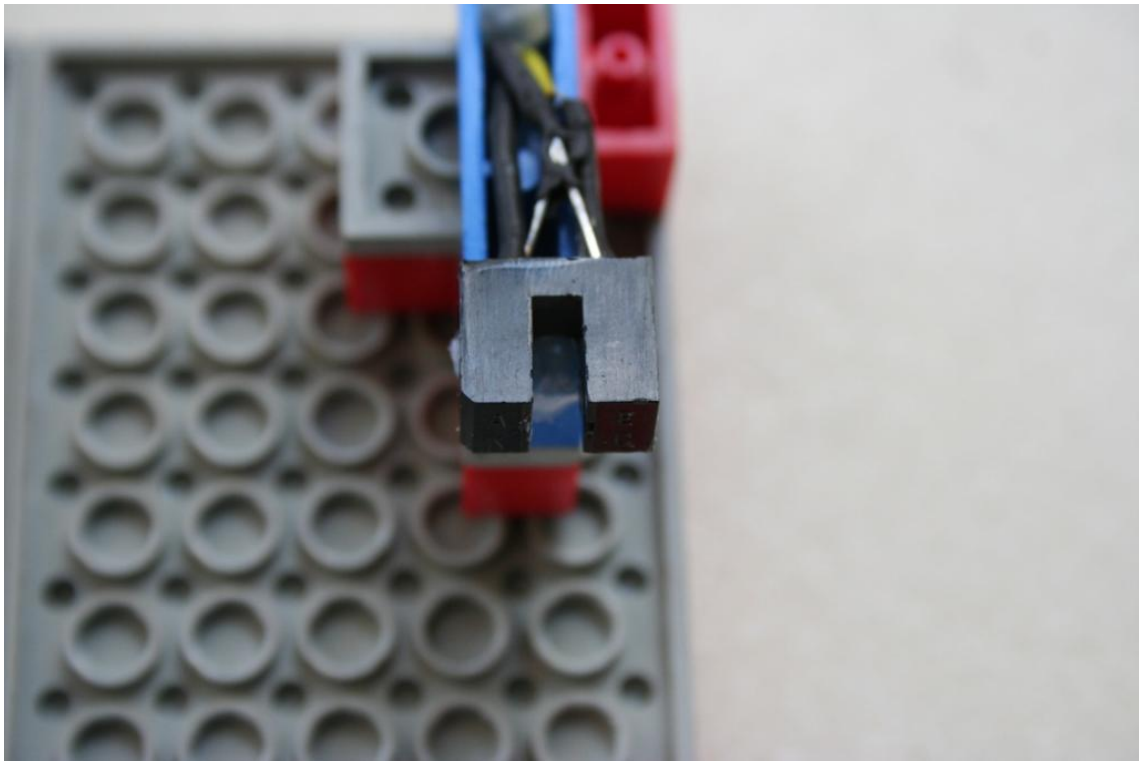

Figura 8: Detalhe de um sensor de interrupção de feixe apoiado na placa superior

Para completar a montagem, as placas mostradas na figura 7 foram colocadas sobre a região dos motores, sendo que estas servem para apoiar a handyboard. Para melhorar a sustentação, formamos uma área ligeiramente maior e montamos vigas nas bordas, para evitar que a placa deslizasse lateralmente. Foram adicionadas travas verticais às laterais da estrutura, e um elástico para evitar que a handyboard deslize. As imagens a seguir mostram a montagem pronta.

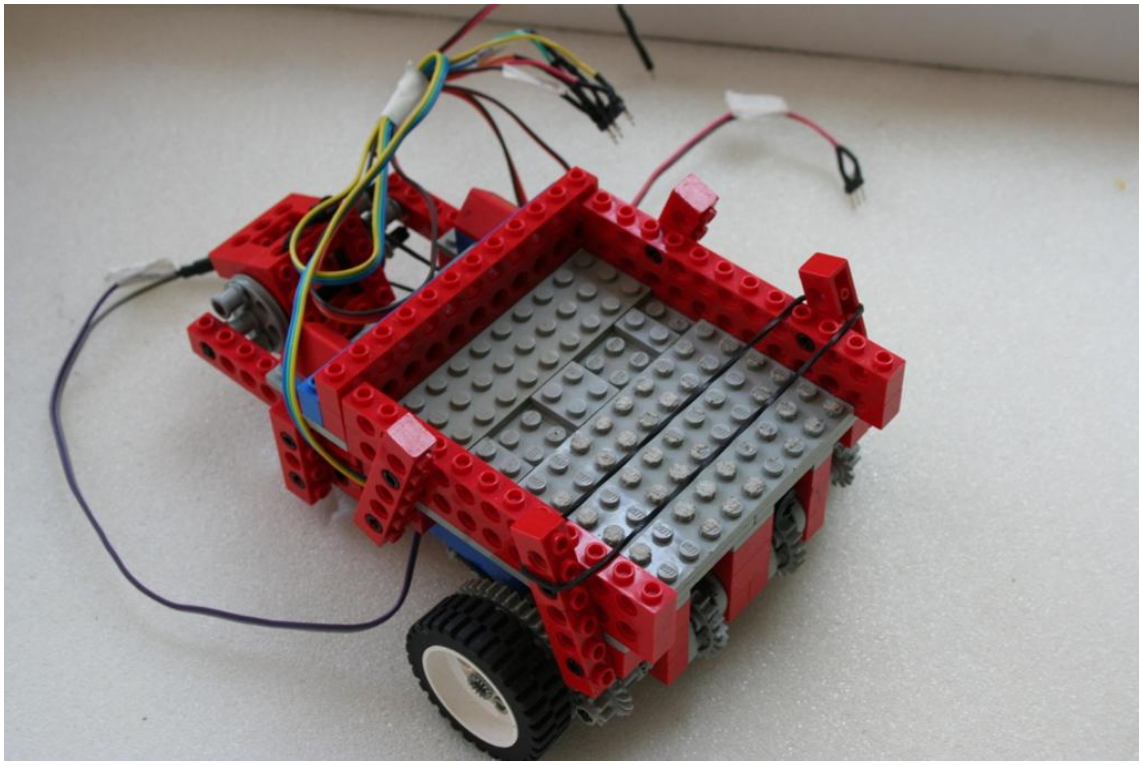

Figura 9: Forma final do Robô sem a handyboard

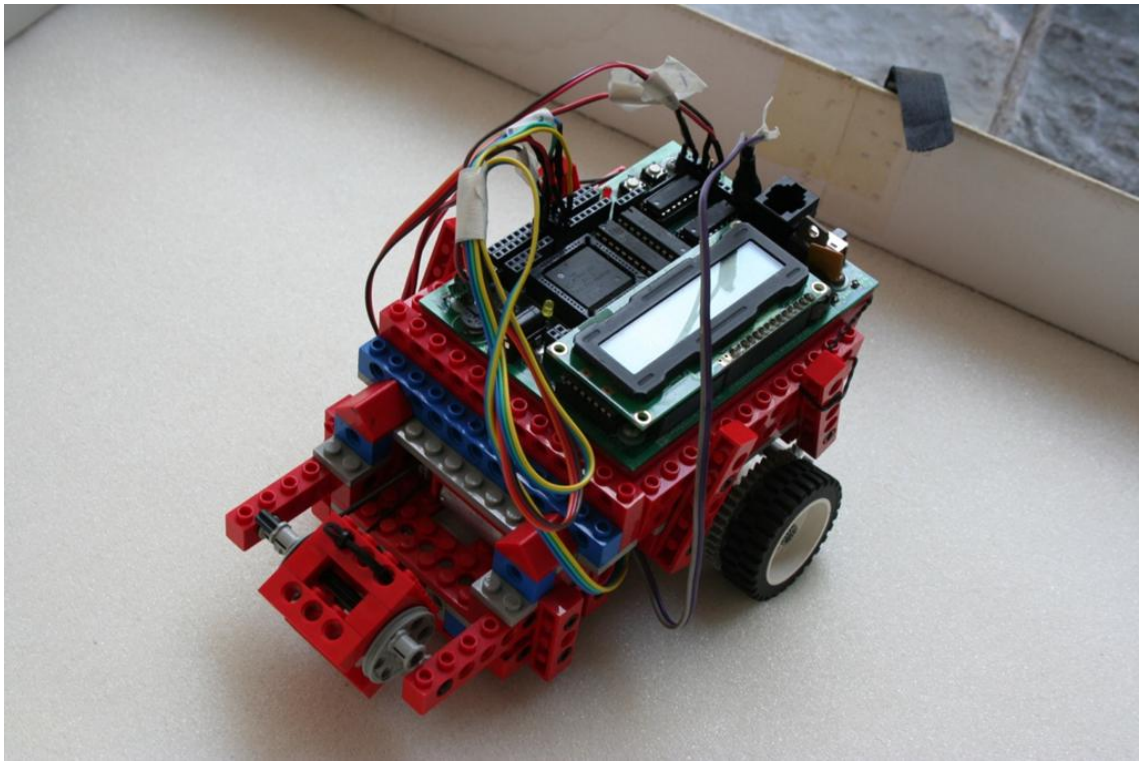

Figura 10: Montagem completa, vista obliqua

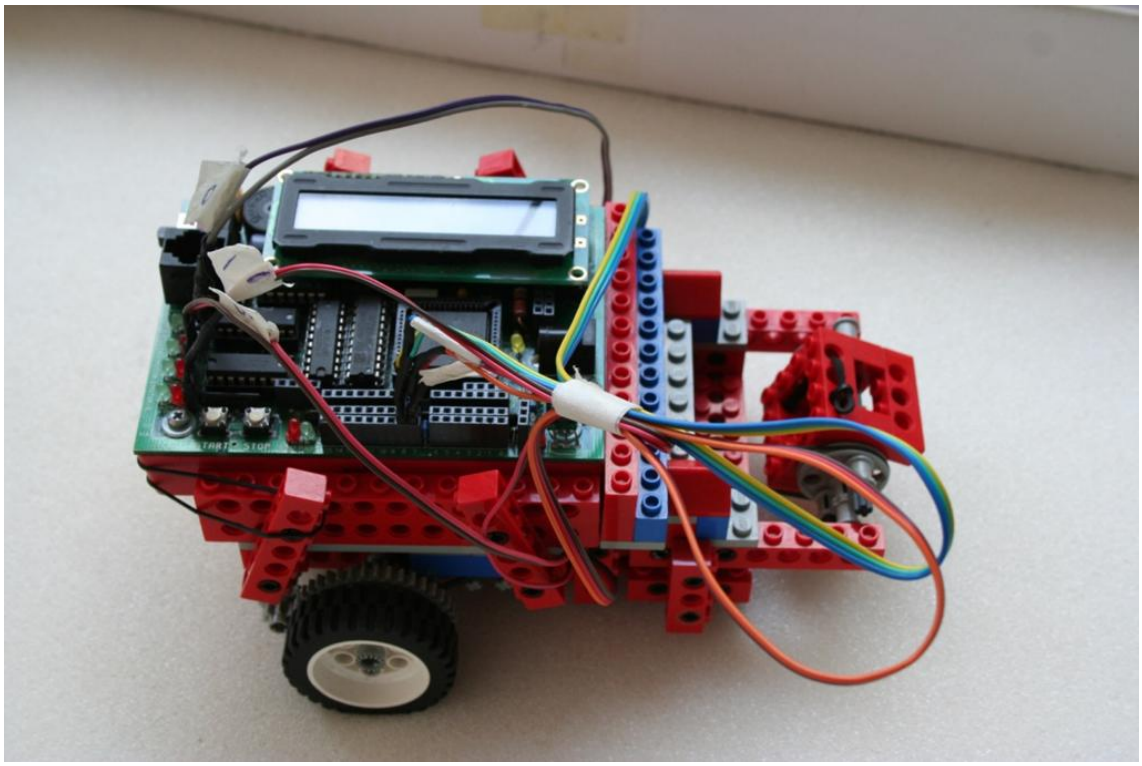

Figura 11: Montagem completa, vista lateral

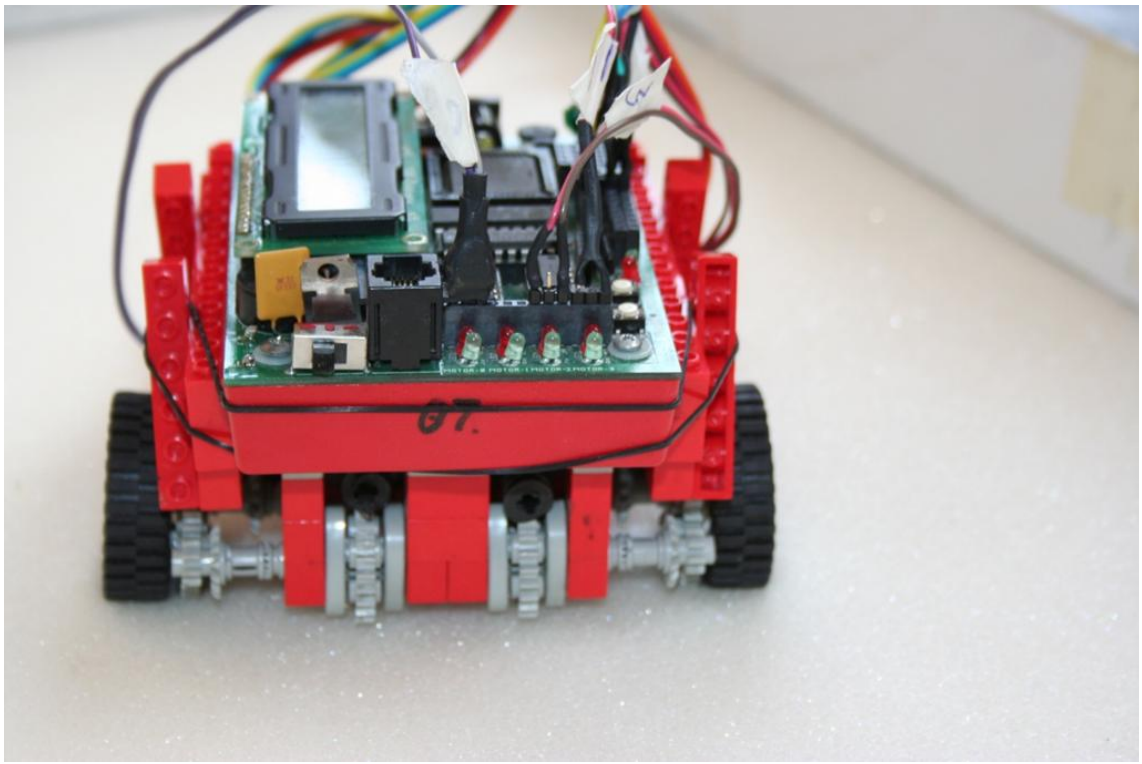

Figura 12: Montagem completa, vista frontal.

Com o robô pronto, faltava apenas nomeá-lo. Lembrando de uma célebre competição fictícia entre veículos exóticos, decidimos prestar nossa homenagem a um clássico programa da época de nossas infâncias. Assim batizamos o nosso robô: “a Carroça a Vapor”.

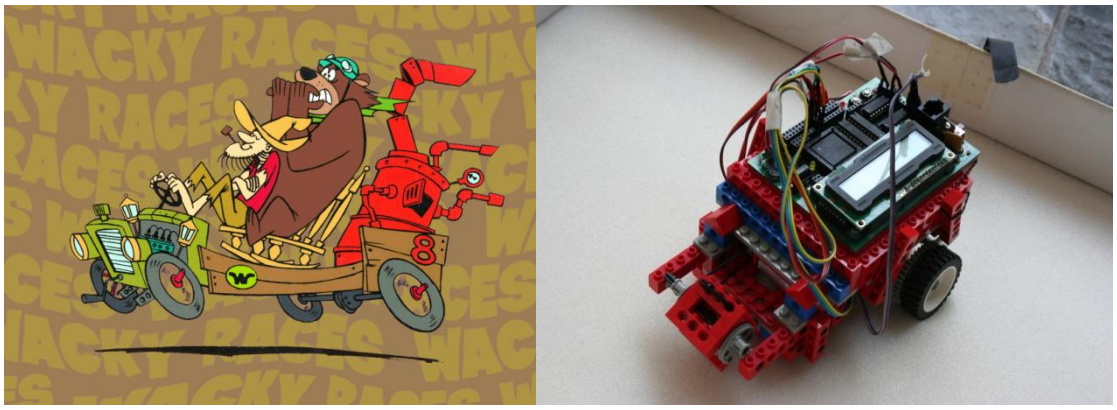

Figura 13: Inspiração para o nome do robô

4. Desenvolvimento do software

Todo o código desenvolvido está transcrito na seção de anexos, com comentários explicativos para fins de documentação.

O objetivo deste robô é desenhar quadrados e círculos. Visando um desenvolvimento modular implementamos duas funções de desenho básicas, uma para traçar arcos e outra para retas. A partir destes dois elementos geométricos podemos compor diversos trajetos e figuras, portanto esta etapa poderá ser necessária em trabalhos posteriores. Visto isso, optamos por uma implementação genérica, de forma que as funções recebam parâmetros que caracterizam o componente a ser desenhado (ângulo e raio para o arco, comprimento para a reta).

Para implementar as duas funções citadas anteriormente, uma terceira ainda mais elementar foi desenvolvida, que calcula a velocidade de uma roda a partir de um shaft-encoder montado nesta. O funcionamento adequado desta rotina depende de sua execução continua, portanto ela é chamada em um processo separado para cada roda, ambos concorrentes entre si e à função principal. A cada meio segundo o valor da velocidade é armazenado em uma variável global. Destaca-se que, como há apenas uma função escrevendo em cada variável compartilhada, não há condições de disputa que exijam controle de acesso.

Assim a função “reta” é apenas um controle proporcional para manter iguais as velocidades das duas rodas e um cálculo da posição para determinar a condição de saída. A função “círculo” usa o mesmo princípio, mas mantendo uma diferença constante entre a velocidade das duas rodas, baseada na diferença entre o raio dos círculos executados por cada uma. O ângulo e o raio são usados para determinar a distância linear que deve ser percorrida. Na implementação atual esta função tem a limitação de desenhar círculos em apenas um sentido.

A implementação também possui uma funcionalidade adicional: permitir que o robô levante e abaixe a caneta. Apesar de não ser exigência deste trabalho, este recurso adicional tem grande impacto na apresentação do robô, além de exigir pouco esforço. De fato, a maioria das alterações necessárias para este objetivo foram em hardware, onde foi necessário adicionar o terceiro motor e projetar uma estrutura de sustentação para a caneta e uma interface com a fonte de tração. As implicações em software foram apenas o acréscimo de poucas linhas, para acionar o motor (para baixar ou levantar), esperar um tempo, e desligá-lo em seguida.

Agora veremos como o robô usa estas ferramentas para executar a tarefa proposta. Desenhar o círculo é uma aplicação direta da função de arcos, sendo o parâmetro “ângulo”

igual a trezentos e sessenta. Os lados do retângulo são também uma aplicação direta da função “reta”, já a curva de noventa graus necessária para fazer a transição entre os lados foi mais desafiadora. A idéia original era que, após desenhar um lado, o robô levantaria a caneta, recuaria uma certa distância e então usaria a função “arco” para girar os noventa graus (sem encostar a caneta na superfície), então faria um novo avanço e abaixaria a caneta para começar o próximo lado. Conforme veremos na discussão dos resultados, houve alguns problemas com este algoritmo.

5. Resultados

No enunciado do trabalho solicita-se que se apresente os gráficos levantados durante a calibração do robô. Porém devido ao uso de um sistema em malha fechada, esta montagem não se utilizou de calibração ou mesmo levantamento de características do motor. Portanto esta seção cobre apenas os desenhos feitos durante os testes e discussões a respeito.

Para avaliar todas as características de interesse, foi elaborado o seguinte roteiro de testes:

- Após ligado, o robô aguarda que o botão “start”seja pressionado;
- O robô desenha uma reta de comprimento 30 cm e aguarda novamente o botão start;
- O robô desenha um quadrado de lado 10 cm e aguarda novamente o botão start;
- O robô desenha círculos de raio 30 cm e para.

A figura 14 mostra duas execuções distintas do trecho do teste que desenha as retas

Figura 14: Duas retas desenhadas pela mesma função

Nota-se que há uma tendência de desvio para a esquerda, mas o experimento mostrou boa repetibilidade.

Na figura 15 apresenta-se o resultado da função de desenhar um quadrado. Devido a resultados diferentes do esperado, foram acrescentadas marcações para indicar o sentido percorrido pelo robô e o número do lado desenhado.

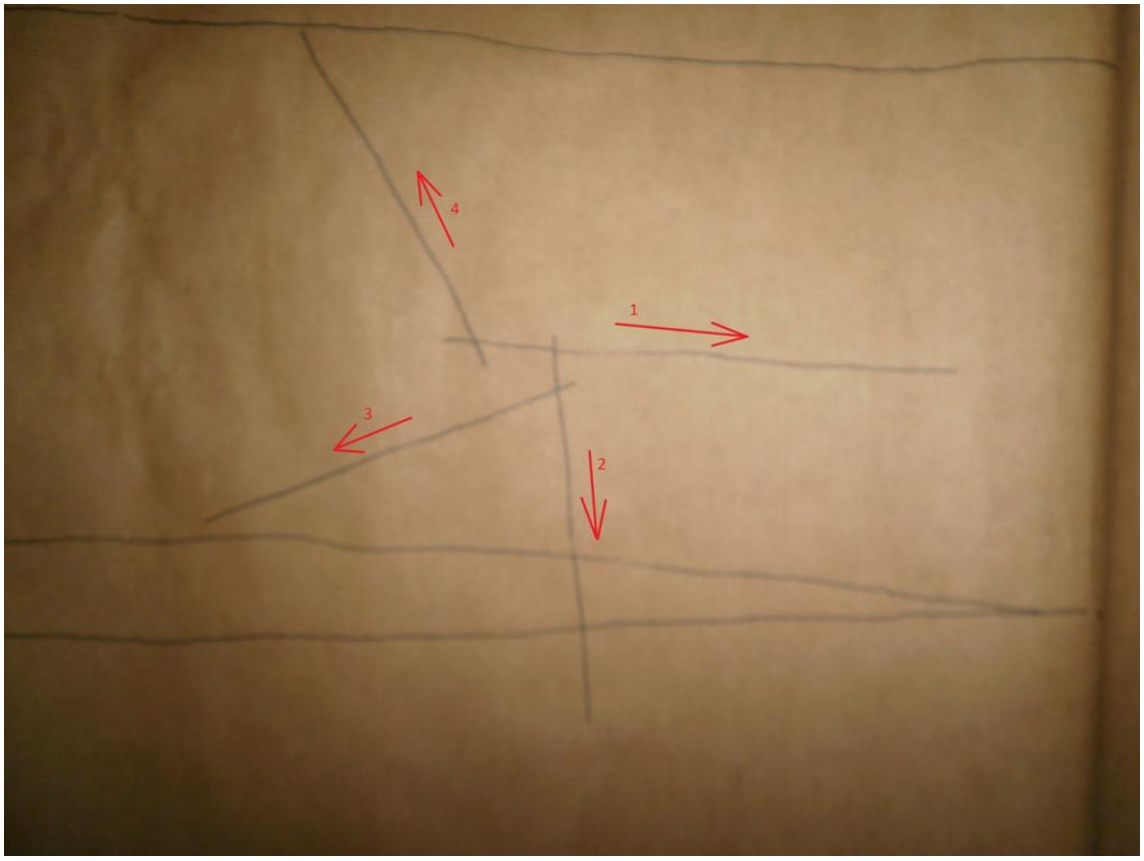

Figura 15: tentativa de desenhar um quadrado

Diferente dos outros experimentos, os resultados deste foram bem abaixo do esperado. Uma razão para isso foi a mudança em relação ao algoritmo inicial. Notamos que havia uma falha com a função “circulo”, de modo que os arcos eram superiores ao ângulo passado como parâmetro. Depois de diversas tentativas infrutíferas de correção, e dada a iminência da apresentação, alteramos a função “retângulo” para fazer o giro de noventa graus em malha aberta. Ou seja, os lados ainda foram desenhados usando a função reta, assim como o recuo e um avanço antes e depois da curva respectivamente, porém o giro foi executado enviando potências de mesmo módulo e sinal contrário para os motores e esperando um tempo para que houvesse o alinhamento com o próximo lado. Este intervalo de atraso foi determinado empiricamente, mas não realizamos uma quantidade de testes suficiente para determinar um valor aceitável. O código deste trecho se encontra na seção de anexos. Observando a figura, nota-se que os lados 1 e 2 estão perpendiculares entre si, assim como os lados 3 e 4. Uma falha clara esta na transição do lado 2 para o 3. Nota-se que o tempo dos recuos está inadequado, pois o começo de cada lado está mais próximo do ponto inicial do

anterior (recuo muito grande). O vídeo disponibilizado no site mostra claramente este problema.

Quanto à distância entre os pontos de contato das retas, houve outro problema além da função de executar giros. Ao terminar um lado, o robô deveria levantar a caneta, girar e então baixá-la para começar o próximo. Uma fonte de erro nesta etapa é que o robô pára de andar para levantar a caneta, mas não para baixá-la (conforme pode ser visto nos vídeos gravados durante os testes). Com isso introduzimos diversas incertezas no sistema, sendo que parar o movimento para manipular a caneta não seria um fator prejudicial nas condições propostas para a disputa.

Nota-se que houve uma atenuação do desvio observado no desenho das retas em relação à figura 14. Como no teste anterior as retas possuíam três vezes o comprimento dos lados desta figura, seria interessante avaliar se este erro seria detectado caso o robô tivesse que desenhar quadrados maiores.

Por fim, apresentamos o resultado da função que desenha círculos na figura 16.

Figura 16: Resultado do teste de desenho de múltiplos círculos

Apesar de a imagem estar fora de foco, percebe-se que há uma translação entre os círculos, porém cada um deles é bastante uniforme.

Conforme citado anteriormente, houve um problema com a função círculo, que fazia o arco de saída apresentar um ângulo muito superior ao de entrada. Além de não conseguir identificar sua origem, também não foi detectada uma relação linear entre o ângulo de entrada e o ângulo de saída, impossibilitando correções por fatores multiplicativos.

6. **Conclusões**

O principal destaque deste trabalho foi o uso de um controle de velocidade com realimentação. Esta característica permitiu que nosso robô apresentasse resultados satisfatórios mesmo com quase nenhuma calibração e “desenvolvimento relâmpago”.

As maiores dificuldades foram na parte de software, porém isso já era esperado, visto que esta etapa foi deixada para o final e o grupo teve problemas em gerenciar o tempo disponível. O funcionamento desta apresentação dependia fortemente da função de desenhar arcos, de modo que gastamos muito tempo para projetá-la e posteriormente para tentar corrigi-la. Outra deficiência foi a falta de dados sobre as características dos motores. O sistema realmente se apresentou bastante robusto em relação a variações de parâmetros, porém quando foi necessário usar um pequeno trecho do sistema em malha aberta os resultados foram desastrosos. Isto ficou bem claro na função de desenhar o quadrado, quando o método de giro idealizado não pode ser aplicado. Além disso, mais conhecimento sobre o sistema poderia explicar outros comportamentos observados, como a tendência que o robô apresentava de virar à esquerda quando desenhava retas ou o fato da transição ente os lados do quadrado apresentar resultados distintos de acordo com o lado, sendo que usamos a mesma função para todos.

As principais lições que tivemos neste trabalho foram:

- O uso de realimentação melhora consideravelmente a insensibilidade do sistema a variações paramétricas;
- Informações sobre o sistema são sempre bem-vindas, principalmente curvas características;
- Devemos conhecer melhor as ferramentas disponíveis. Por exemplo, implementamos manualmente funções de aquisição contínua de dados para o encoder, sendo que o próprio IC possui rotinas prontas para esta função, disponíveis como device drivers para a handyboard;
- É fundamental aprender a distribuir bem o tempo entre as etapas do trabalho.

7. Anexos

7.1. Primeiro Código desenvolvido para o programa

```
/*
 Código desenvolvido pelo Grupo 07 para o primeiro trabalho de introdução a
robótica

 O robo deve ser programado para desenhar circulos e quadrados. Assim,
construimos duas funções
 básicas:
 -> Circulo, permite ao robô desenhar arcos de circunferência, recebendo como
parâmetro o raio e o ângulo desejados.
 -> Reta, faz com que o robô desenhe uma linha reta cujo o comprimento é
recebido como parâmetro.

 Nosso robô também possui sensores movimento angular, e foi desenvolvida a
função "emissor" para  monitorá-los. Como esta medida é contínua, esta função
será executada concorrentemente ao  programa principal, e depositará seus
dados em uma variável global.
 */


 /****** Considerações *****
 1. Ao longo de todo o programa, quando usamos números para representar algo
associado a um lado ou outro do robô, adotaremos a seguinte convenção:
```

```
 0 => direita
 1 => esquerda
 */

 /****** Variáveis globais ******/

//variáveis para a comunicação entre os processos de leitura do sensor e o
programa principal.
float vel[2];

// valores de potência serão armazenados em variáveis globais, também para
fins de comunicação
int pot_esq,pot_dir;

void emissor(int lado)
{/*
       Esta função monitora as medidas de um sensor de interrupção de feixe.
       A cada iteração mede-se o valor atual da contagem em um dado instante e
o valor da contagem meio segundo depois. O módulo da diferença entre estas
medidas, multiplicado pelo raio fornece a velocidade da roda.
       É importante destacar que o esta função não diferencia o sentido de
rotação. O conhecimento sobre esta variável será totalmente derivado dos
comandos enviados para acionar o motor.*/

    int count_atual = 0;
      int count_init = 0;

    while(1)
      {
            //Lê a contagem do encoder no início do intervalo
        count_init = read_encoder(lado);

            //gerando um atraso de meio segundo
        sleep(0.5);

            //Lê a contagem do encoder ao final do intervalo
            count_atual = read_encoder(lado);

            /*determinando da velocidade*/
            vel[lado] = sqrt((0.01*((float)(count_atual - count_init)))^2);

    }
}


void circulo(float raio, float angulo)
{
       /*
       Função para desenhar arcos de circunferência.
       A idéia é aplicar potências distintas em cada motor para gerar a
circunferência. A diferença entre as potências será determinada pelo raio
desejado, enquanto o ângulo determinará a distância que o robô deverá
percorrer. Para simplificar, vamos manter o motor esquerdo a uma potência
constante e reduzir a do direito.
       Destacamos que esta abordagem implica em desenhar circulos sempre em um
mesmo sentido, que neste caso será horário. (consideramos que o parâmetro raio
será sempre positivo. No entanto não acrescentamos código para fazer esta
verificação. O ângulo será recebido em graus e convertido a uma distância
linear correspondente. A distância percorrida é a determinada pela velocidade
do ponto central do eixo de tração multiplicada pelo raio das rodas
       */

       //cálculo da distância a percorrer a partir do ângulo
    float dist = 3.14159*raio*angulo/180.0;

       //variável que armazena a distância percorrida
```

```
    float distPerc = 0.0;

        //A potência do motor esquerdo será mantida em valor constante
    pot_esq = 50;

       /*A potência do motor direito é determinada a pelo raio desejado*/
    pot_dir = (int)(50.0*((raio-0.06)/(raio+0.06)));

       //acionando o terceiro motor para abaixar a caneta.
       motor(1, -55);

       /***** loop para desenhar o círculo ******/
    while(1)
      {
        //acionando dos motores
       motor(3,pot_dir);
       motor(0,pot_esq);

            /****** controle da velocidade ******/

            /*como a potência do motor esquerdo é "constante", o erro será
avaliado pelo desvio do motor direito em relação a este parâmetro*/

            //verificando se a velocidade do lado direito está muito alta
        if(vel[0] > vel[1]*(raio-0.06)/(raio+0.06))
        {// reduzindo a potência do motor direito, porém nunca abaixo de 20%
            if(pot_dir > 20)
             pot_dir--;

                  //se a potência do motor direito já está muito baixa,
aumenta a do lado esquerdo
                  else
             pot_esq++;
        }

            //verificando se a velocidade do lado direito está muito baixa
            if(vel[0] < vel[1]*(raio-0.06)/(raio+0.06))
        {// aumentando a velocidade do lado direito
            ++pot_dir;

               /*
               note que agora não nos preocupamos em avaliar limites para a
potência do motor direito, pois se ela se desviar demais, será corrigida pela
verificação anterior
                  */
        }
        sleep(0.001);

            //atualizando a distância percorrida
        distPerc += ((vel[1]+vel[0])/2.0)*0.001;

            //quando a distância percorrida superar o limite estabelecido,
encerra o loop
            if(distPerc >= dist)
          break;
    }

       //levanta a caneta
    motor(1, 55);
    sleep(1.0);
    motor(1,0);
}

void reta(float distancia)
```

```
{
	/*
	função para desenhar uma linha reta de comprimento variável.
	Esta função se baseia em um controle de da velocidade das rodas. A cada
iteração avalia-se a velociade de uma roda em relação a outra, e qualquer
desvio gera provoca uma variação de 1% na potência de ambos os motores. O
valor de referência para a potência dos motores é 30%.

	Note que a função aceita também valores negativos do comprimento,
indicando neste caso que o robô andará "de ré"
	*/

    /*dist é uma variável auxiliar, para armazenar o módulo do comprimento a
ser percorrido*/
    float distPerc, dist;

	//inicializa a distância já percorrida
	distPerc = 0.0;

	/****** determinando dos parâmetros em função do sentido do movimento
******/
    if(distancia >= 0.0)
      {
        pot_esq = 30;
        pot_dir = 30;
        dist = distancia;
    }
    else
      {
        pot_esq = -30;
        pot_dir = -30;
        dist = -distancia;
    }

	/****** loop para desenhar a reta ******/
    while(1)
      {
            //acionando os motores com o valor calculado para a potência de
cada lado.
        motor(3,pot_dir);
        motor(0,pot_esq);

        if(vel[1] > vel[0])
          {/*
                se a velocidade da roda direita é maior que a da esquerda,
verifica o desvio em relação ao valor de referência (30%)*/
            if(sqrt((float)pot_dir^2.0) >= 30.0)
              {/*
                    se o módulo de potência é maior do que 30%, aplica a
correção adequada baseado no sentido do movimento */
                if(distancia >=0.0)
                        {
                    pot_dir--;
                }
                        else
                        {
                    pot_dir++;
                }
            }
            else
            {/*
                    se o módulo de potência é menor do que 30%, aplica a
correção adequada baseado no sentido do movimento */
                if(distancia >=0.0)
                        {
                    pot_dir++;
```

```
                }
                        else
                        {
                    pot_dir--;
                }
            }
        }
        if(vel[0] > vel[1])
          {/*
                 se a velocidade da roda esquerda é maior que a da direita,
verifica o desvio em relação ao valor de referência (30%) */
            if(sqrt((float)pot_esq^2.0) >= 30.0)
              {/*
                se o módulo de potência é maior do que 30%, aplica a correção
adequada baseado no sentido do movimento */
                if(distancia>=0.0)
                        {
                    pot_esq--;
                }
                        else
                        {
                    pot_esq++;
                }
            }
            else
            {/*
              se o módulo de potência é menor do que 30%, aplica a correção
adequada baseado no sentido do movimento */
                if(distancia>=0.0)
                        {
                    pot_esq++;
                }
                        else
                        {
                    pot_esq--;
                }
            }
        }
        sleep(0.1);

             //atualiza o valor da distância já percorrida
        distPerc += vel[1]*0.1;

        printf("Reta: %f\n",distPerc);

             //quando a distância percorrida superar o limite estabelecido,
encerra o loop
        if(distPerc >= dist)
          break;
    }

       //desligando os motores
       motor(0,0);
    motor(3,0);
}

void retangulo(float a, float b)
{
       /*
       Esta função desenha um retângulo, recebendo como parâmetros o
comprimento de dois de seus lados. Cada lado é desenhado fazendo uma chamada à
função "reta". Após desenhar um lado, é ordenado ao robô executar um récuo
equivalente à distância entre o centro do eixo de tração e o ponto de contato
da caneta, um giro de noventa graus (usando a função "circulo", implementada
neste mesmo arquivo), e um avanço de mesmo comprimento do récuo (que também é
o comprimento do raio do arco).
       */
```

```
      /****** Desenhando o lado 1 ******/

      //encosta a caneta no papel
      motor(1,-55);
    reta(a);

      //levanta a caneta
    motor(1, 55);
    sleep(1.0);
    motor(1,0);

      //Alinhando com o próximo lado
    reta(-0.16);
    circulo(0.16, 90)
    reta(0.16);

      /****** Desenhando o lado 2 ******/

      //encosta a caneta no papel
      motor(1,-55);
    reta(b);

    //levanta a caneta
    motor(1, 55);
    sleep(1.0);
    motor(1,0);

      //Alinhando com o próximo lado
    reta(-0.16);
    circulo(0.16, 90)
    reta(0.16);

      /******Desenhando o lado 3 ******/

      //encosta a caneta no papel
    motor(1,-55);
    reta(a);

      //levanta a caneta
    motor(1, 55);
    sleep(1.0);
    motor(1,0);

      //Alinhando com o próximo lado
    reta(-0.16);
    circulo(0.16, 90)
    reta(0.16);

      /****** Desenhando o lado 4 ******/

      //encosta a caneta no papel
    motor(1,-55);
    reta(b);

      //levanta a caneta
    motor(1, 55);
    sleep(1.0);
    motor(1,0);
}

//função principal
void main()
```

```
{
	/*
	Para fim de registro, definimos que nosso robô desenharia uma linha
reta, um quadrado e um circulo, nesta ordem. Para que o robô pudesse ser
deslocado entre cada uma destas etapas (para evitar a superposição de
desenhos, por exemplo), ele espera que o botão "start" da handyboard seja
pressionado antes de começar a desenhar.
	Note que os parâmetros dos desenhos (comprimento da reta, dimensões do
quadrado e do círculo) são definidos em no código, portanto sua alteração
exige que se refaça o download do código para a handyboard.
	*/

	//habilita a porta onde conectamos os sensores para o uso da função
"read_encoder()"
    enable_encoder(1);
    enable_encoder(0);

	//cria os processos para leitura continua de cada sensor ótico
    start_process(emissor(0));
    start_process(emissor(1));

	//aguarda que o botão "start" da handyboard seja pressionado
    while(!choose_button()){}
    while(choose_button()){}

	/****** desenhando uma linha reta *****/

	//encosta a caneta no papel
    motor(1,-55);

	//desenha uma reta de 30 cm
	reta(0.30);

	//levanta a caneta
    motor(1,55);
    sleep(1.0);
    motor(1,0);

	//aguarda que o botão "start" da handyboard seja pressionado
    while(!choose_button()){}
    while(choose_button()){}

	/****** desenhando um quadrado com 10 cm de lado ******/
    retangulo(0.10,0.10);

	//aguarda que o botão "start" da handyboard seja pressionado
    while(!choose_button()){}
    while(choose_button()){}

	/***** desenha um círculo de raio 30 cm ******/
    circulo(0.3,360.0);
}
```

7.2. Código para desenho do retângulo utilizado

```
void retangulo(float a, float b)
{
    motor(1,-55);
    motor(0,0);
    motor(3,0);
    reta(a);
    motor(0,0);
```

```
    motor(3,0);
    motor(1,55);
    reta(-0.16);
    motor(0,-100);
    motor(3,100);
    sleep(0.5);
    motor(0,0);
    motor(3,0);
    reta(0.16);
    motor(3,0);
    motor(0,0);
    motor(1,-55);
    reta(b);
    motor(0,0);
    motor(3,0);
    motor(1,55);
    reta(-0.16);
    motor(0,-100);
    motor(3,100);
    sleep(0.5);
    motor(0,0);
    motor(3,0);
    reta(0.16);
    motor(3,0);
    motor(0,0);
    motor(1,-55);
    reta(a);
    motor(0,0);
    motor(3,0);
    motor(1,55);
    reta(-0.16);
    motor(0,-100);
    motor(3,100);
    sleep(0.5);
    motor(0,0);
    motor(3,0);
    reta(0.16);
    motor(3,0);
    motor(0,0);
    motor(1,-55);
    reta(b);
    motor(0,0);
    motor(3,0);
    motor(1,55);
}
```