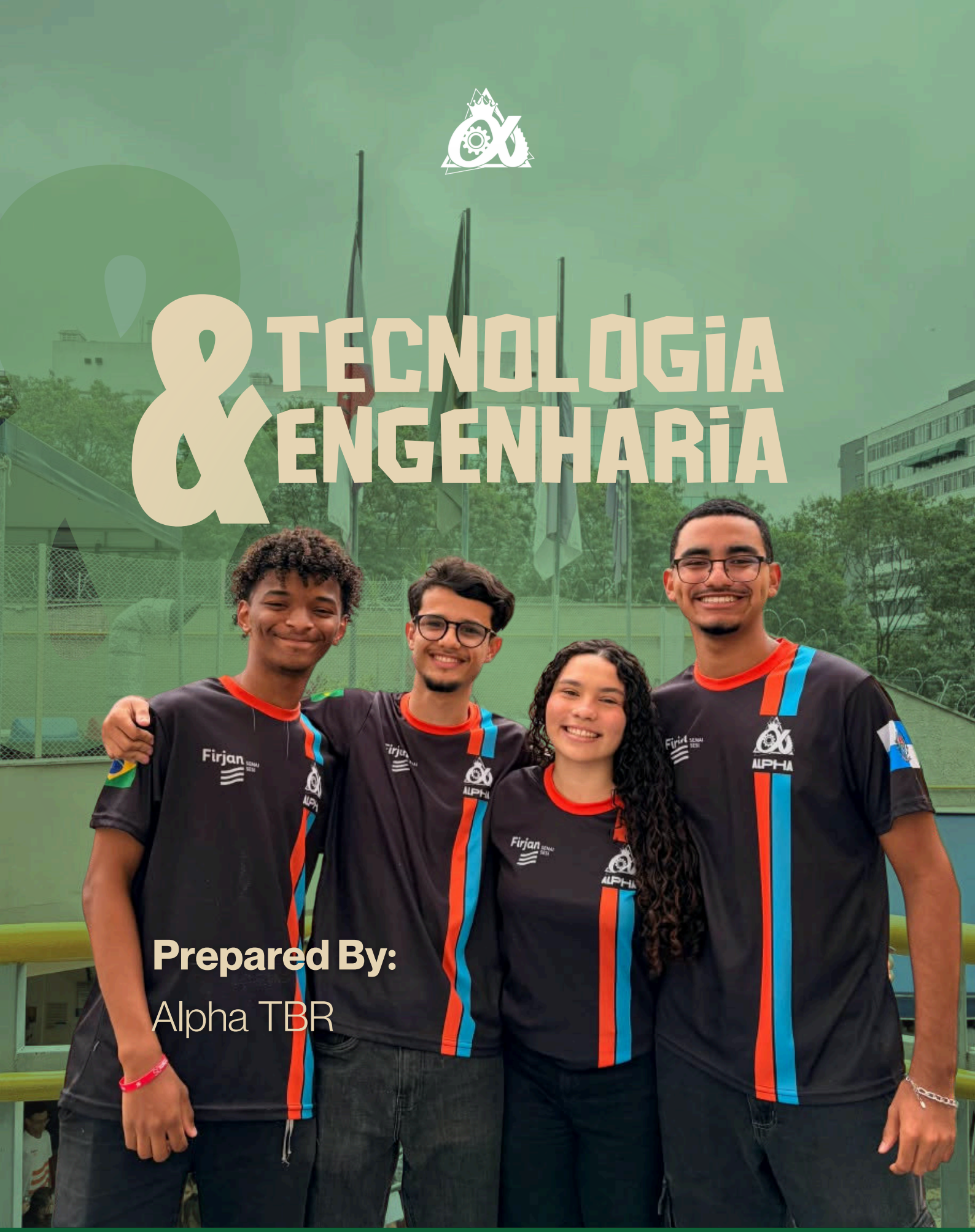
& TECNOLOGIA & ENGENHARIA



Prepared By:

Alpha TBR

TBR

Firjan SENAI SESI

SUMÁRIO

Sumário	1
Apresentação do time	2
Análise de documentos	3
Mentorias	4
Protótipos	5 e 6
Studio 2.0	7
Estratégias	8
Sistemas utilizados	9 e 10
Design	11
Desafios	12
Plano de Contingência	13
Inovações	14
Desafios na Programação	15
Classe PID	16
Movimentação	17
Giro e rotas	18 e 19
Anti-windup e deadband	20 e 21
Softwares	22

ORGANIZAÇÃO
da equipe



MARCO ANTÔNIO
Técnico

Professor de Física no SESI e técnico da equipe. Responsável pela **coordenação** geral da equipe, fornecendo **suporte na construção do robô** e ajudando em **questões técnicas do projeto.**



FABIANE MARTINS
Técnico

Professora de Química no SESI e técnica da equipe. Encarregada da **gestão e liderança**, estabelecendo metas, mantendo a **motivação da equipe** e oferecendo **suporte em aspectos técnicos do projeto**.



FELIPE BARRETO
Auxiliar Técnico

Ex-aluno e competidor, ele contribui de forma essencial à equipe, aplicando o conhecimento técnico adquirido no curso de engenharia elétrica do **CEFET-RJ.**

ALPHA TBR

EQUIPE

MÉRITO CIENTÍFICO	TECNOLOGIA E ENGENHARIA	ORGANIZAÇÃO E MÉTODO	GERENCIAMENTO
Ana Luíza	Danillo Nascimento	Giovanna Lagoa	Fabiane Martins
Pedro Douglas	João Pedro	Sarah Costa	Marco Antonio
Matheus Cardoso	Karol Balbino		Felipe Barreto
Sofia Cordeiro	William dos Santos		

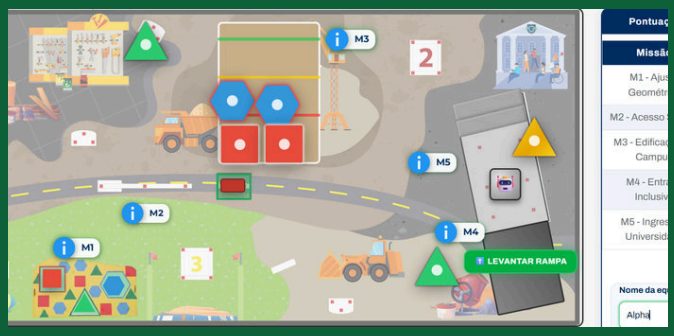
ANÁLISE DE *documentos*

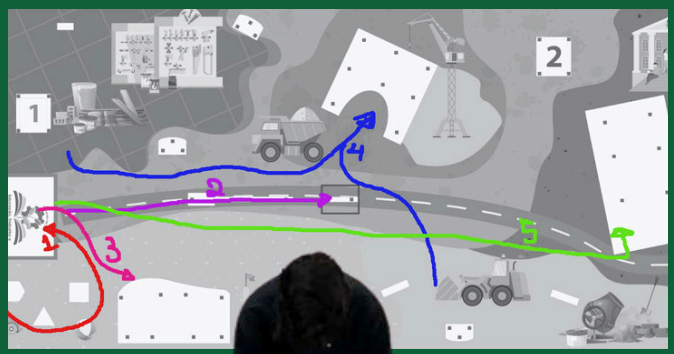
Leitura de manuais

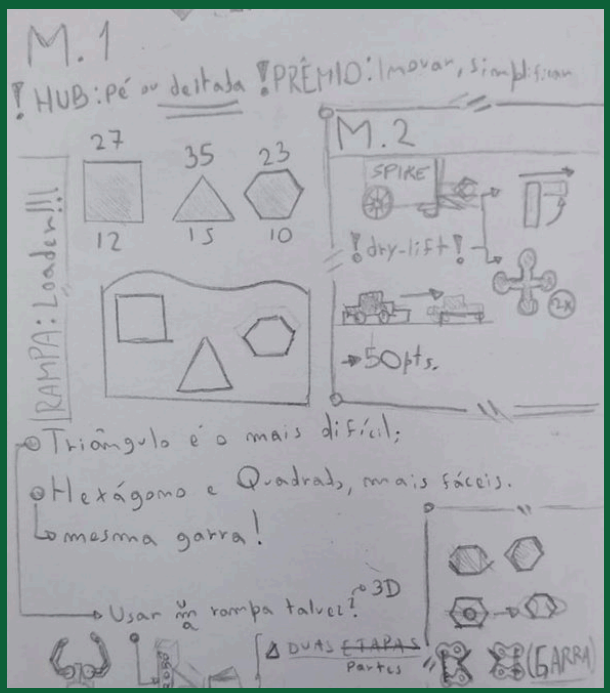
Realizamos a leitura de manuais e a partir disso começamos a notar estratégias que poderíamos utilizar durante o processo criativo.

Ferramentas do TBR

Aproveitamos para analisar as ferramentas criadas pelo site TBR, como o tapete online e o tapete de estratégias.









Início das montagens

MENTORIAS

Ex-membros da Alpha: Alpha Legacy



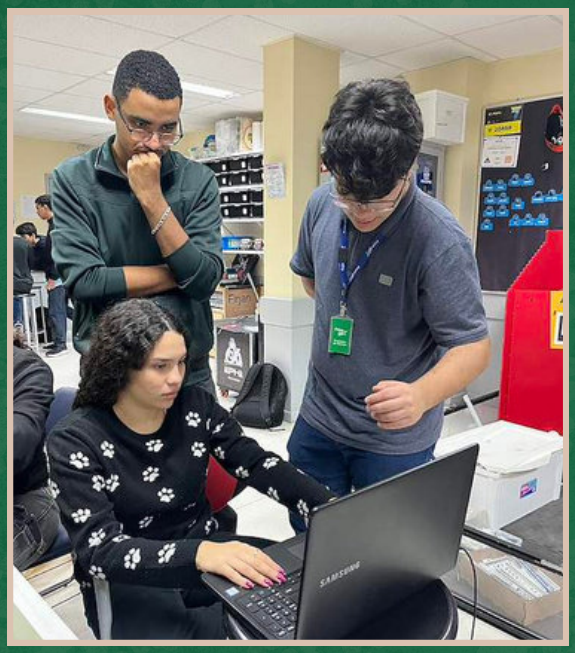
Natan Santiago



Luccas Oliveira



Caio Carvalho
e Gabriel Paiva



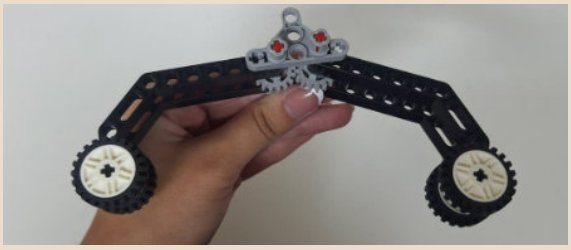
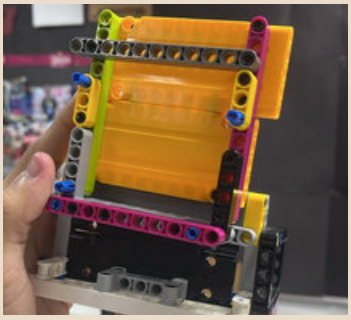
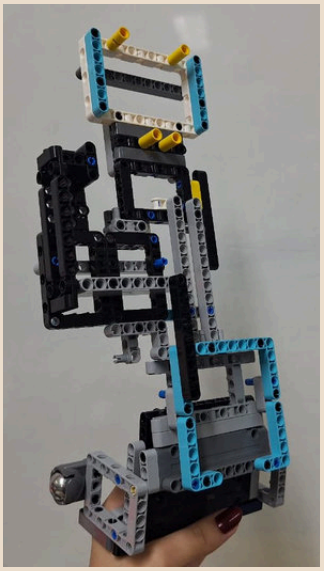
Rafael Tanus

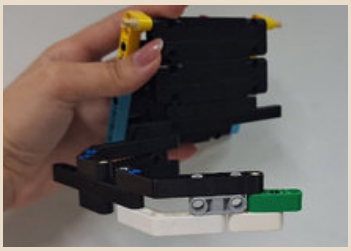
PROTÓTIPOS

DURANTE A TEMPORADA...

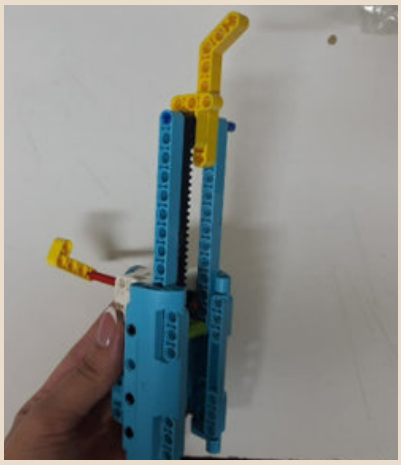
8	7	3	4
missão 1	missão 3	missão 4	missão 5

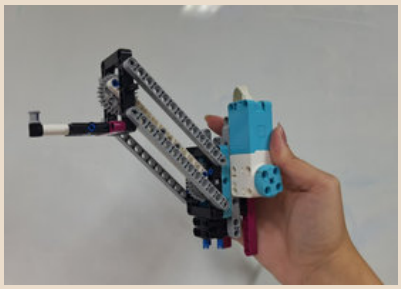
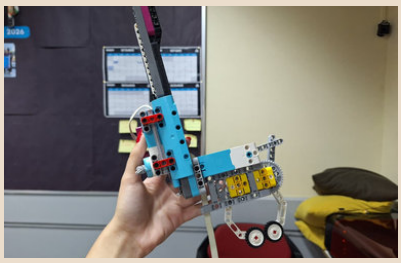
Missão 1: Ajuste Geométrico



Missão 3: Edificação de Campus

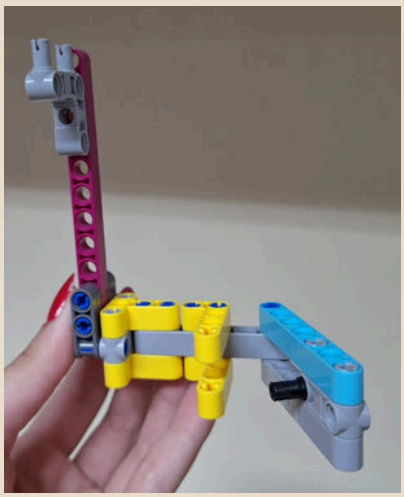


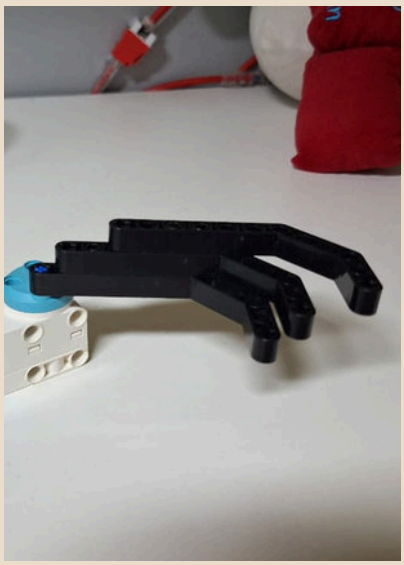



PROTÓTIPOS

Missão 4: Entrada Inclusiva





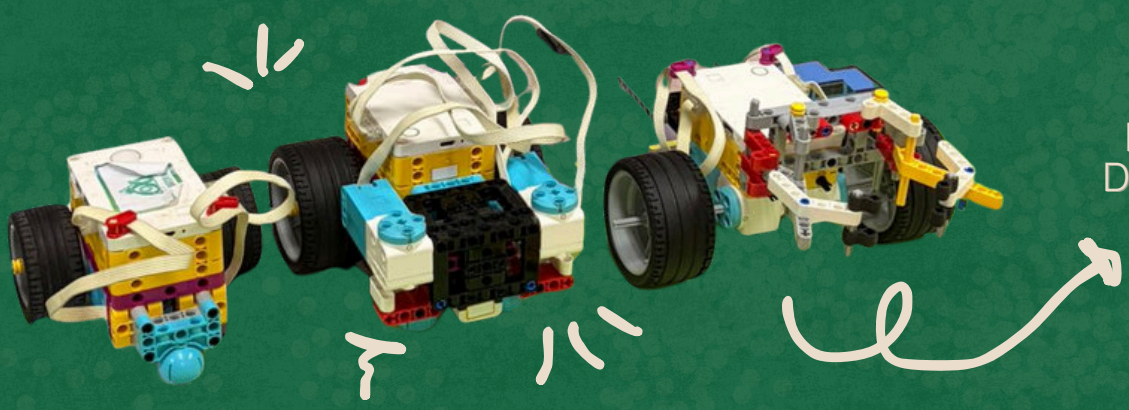


Missão 5: Ingresso à Universidade









PROTOTIPAGEM
DA BASE DO ROBÔ

STUDIO 2.0

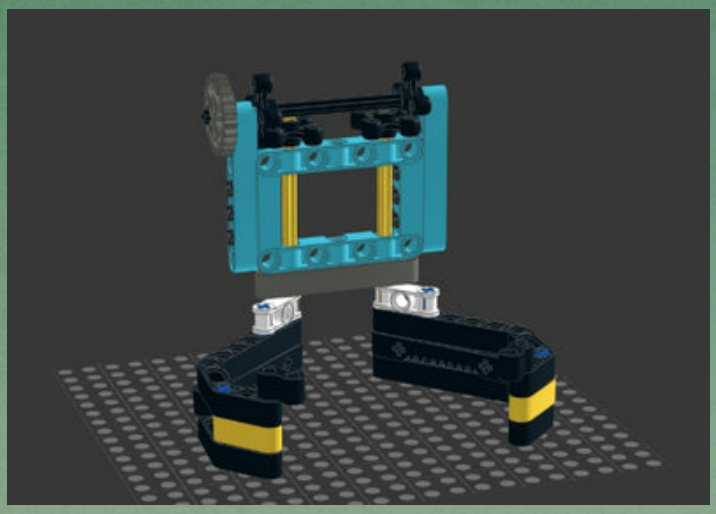
modelo 3D

Modelagem em 3D

Utilizamos o Studio 2.0 para criar e visualizar diferentes mecanismos antes de montá-los. Isso permite uma construção segura, eficiente e a prevenção de erros estruturais.

Estabilidade

Desta forma, desenvolvemos nossos protótipos avaliando o risco, a estabilidade e a resistência de cada mecanismo.



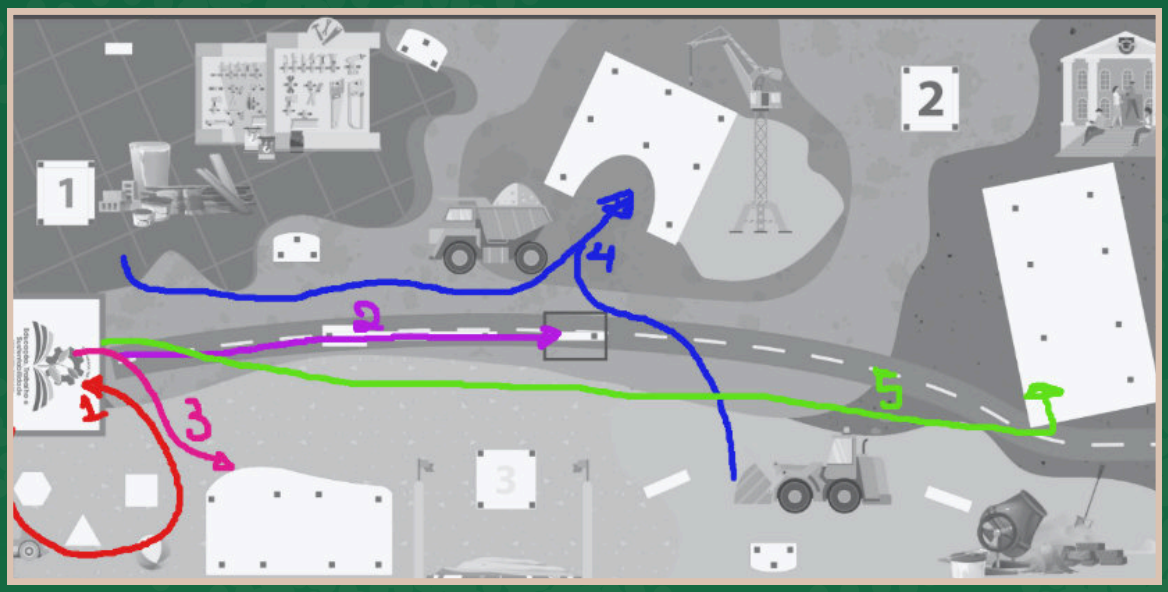


ESTRATÉGIAS
pensadas

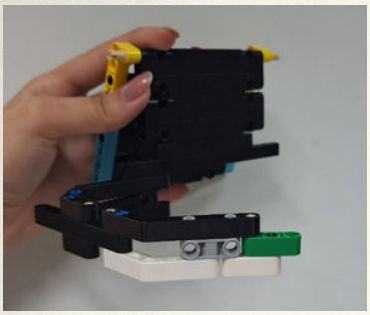
Rota única: sem objetos estratégicos



Rota final: com objetos estratégicos

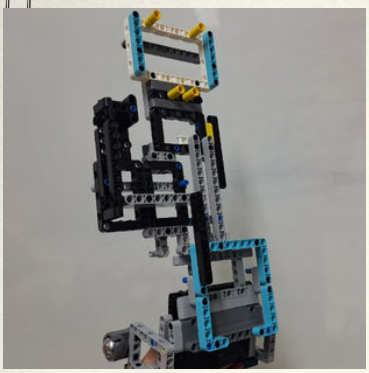


SISTEMAS
utilizados



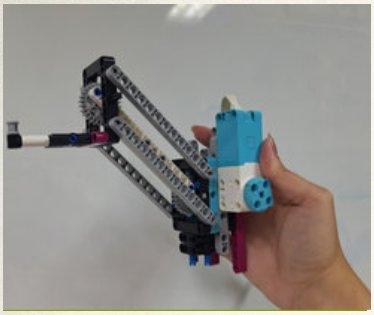
GARRA ESTÁTICA

Para pegar os objetos da missão 1, criamos uma garra estática, com a ponta capaz de encostar no MDF e empurrar as formas geométricas para base inicial.



POSICIONADOR GEOMÉTRICO

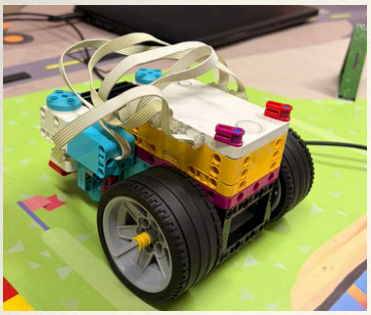
Elaboramos um sistema capaz de colocar as três peças da missão 1 no local apropriado de uma única vez. Garantindo assim uma economia de tempo para realizazr as outras missões.



SISTEMA DE ELEVAÇÃO

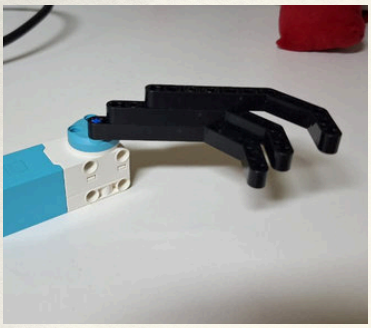
Para missão 3, montamos uma espécie de garra universal, capaz de pegar qualquer formato. Para elevação, criamos um sistema baseado na emplhadeira de indústrias e mercados.

SISTEMAS *utilizados*



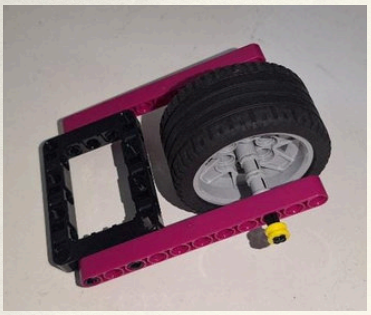
BASE SÓLIDA

Projetamos uma base sólida, capaz de empurrar o carrinho da missão 2 com o próprio robô, sem precisar desenvolver um mecanismo para isso, de uma maneira simples e eficiente.



ALAVANCA SIMPLES

Desenvolvemos uma alavanca horizontal, com o intuito de acionar o motor e ela projetar força sobre a rampa e assim conseguir abaixa-la.



PESO DE SUSTENTAÇÃO

Notamos que o maior desafio para subir a rampa era a distribuição desigual de peso do robô. Pensando nisso, utilizamos um pneu ligado a duas barras para criar um mecanismo de sustentação.

DESIGN do robô

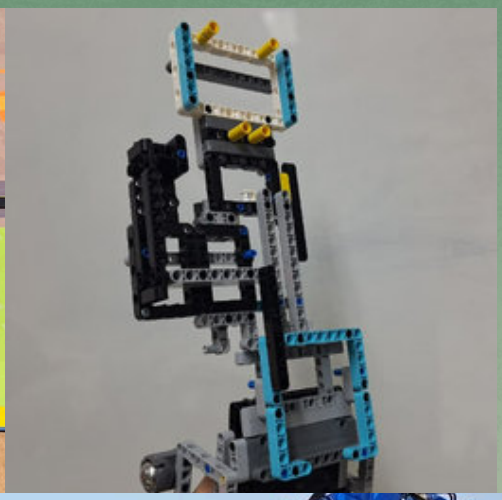
Personalizado

Durante todo o processo criativo, desencolvemos algo compacto, capaz de ser eficiente e rápido. Para isso, apostamos em sistemas personalizados para cada missão.

Simplicidade

Nossa outra aposta, foi criar mecanismos mais simples, que atendesse exatamente nossa necessidade, como por exemplo a alavanca da missão 4.





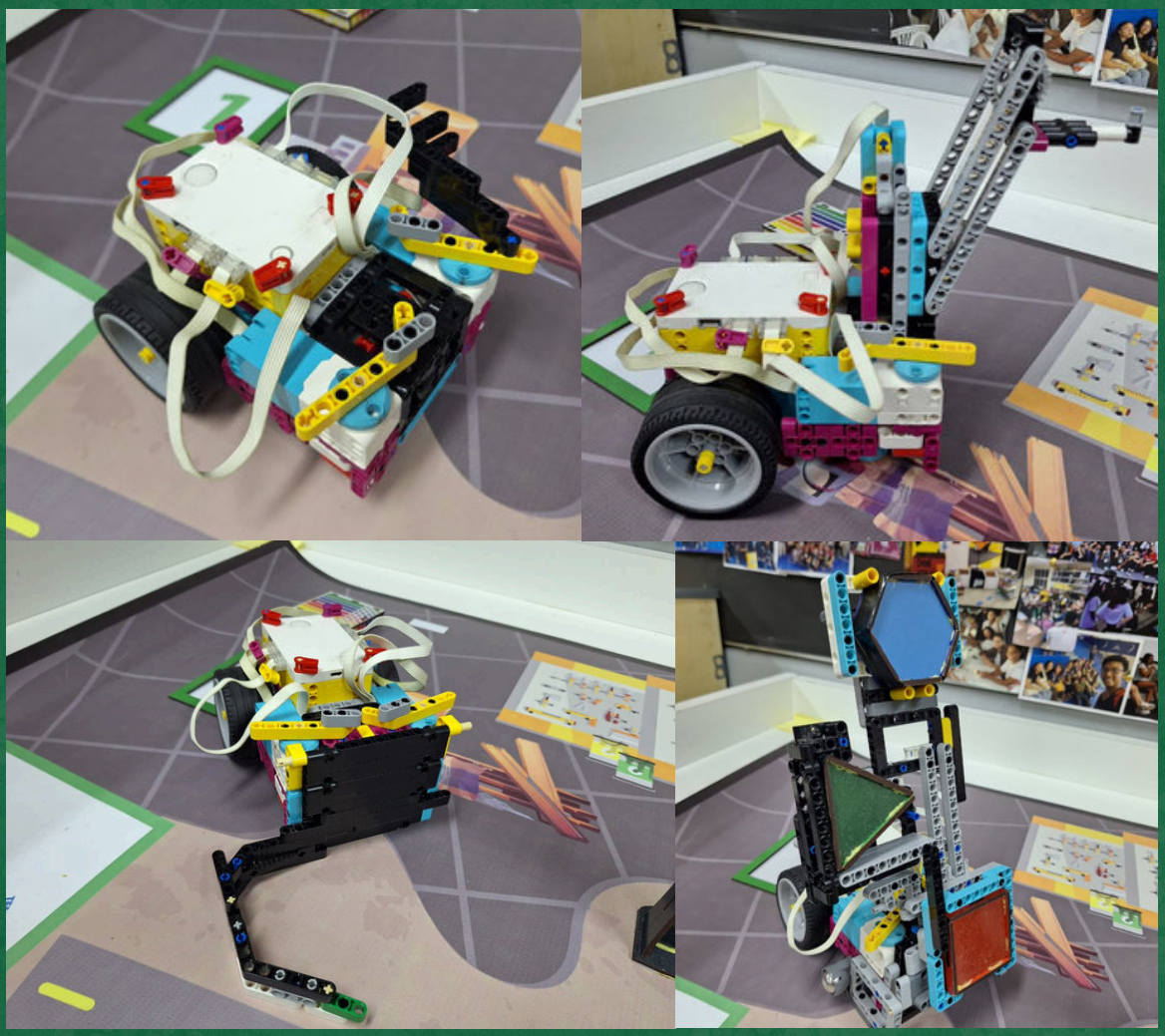




Inspirações para o nosso robô

DESAFIOS
e soluções

Desenvolver sistemas diferentes para um único robô



Solução: Criamos mecanismo personalizados e de fácil remoção, para economizar tempo e garantir precisão.

PLANO DE *contingência*

Contingência Técnica						
AÇÃO	RISCOS	PROBABILIDADE	IMPACTO	NÍVEL DE RISCO	PLANO DE CONTINGÊNCIA	RESPONSÁVEL
TE	Pouca aderência do material a coleta dos objetos da missão 1	4	2	8	Adicionamos uma peça de borracha na ponta da garra, garantindo maior superfície de contato e menos risco de pouca aderência	TE
	Desvio da rota caso ocorra uma penalidade	2	3	6	Aplicação de sensor de força para uma rota alternativa	TE
	Robô errar a rota pelo baixo nível de bateria	3	3	9	Trocar a bateria antes do round oficial, assegurando uma rota eficiente. (Conferir o nível da bateria que será colocada)	TE
	A roda do robô não consegue sustentar a subida dele durante a missão 5, cujo objetivo é realizar o estacionamento.	2	3	6	Utilizar uma roda maior, como a do EV3, para garantir maior precisão e estabilidade durante a subida.	TE
	Errar o posicionamento do robô no lançamento	3	3	9	Realizar a montagem de uma régua para maior precisão na hora do lançamento.	TE

PROBABILIDADE (qual a chance de acontecer?)		IMPACTO (o quanto dói se acontecer?)	
1	Raro (quase impossível)	1	Resolvemos em 10 minutos. Ninguém percebe.
2	Pouco provável: pode acontecer, mas não é o padrão	2	Atrapalha um pouco, gera retrabalho, mas não para a equipe.
3	Possível: 50/50. Acontece com frequência no mercado	3	Precisamos parar para resolver. Pode atrasar cronograma ou perder verba parcial.
4	Provável: Grande chance de acontecer	4	Perda significativa de dinheiro ou reputação. A equipe sente o golpe.
5	Quase certo: É praticamente garantido que vai ocorrer.	5	A equipe para de funcionar ou não consegue competi (ex: sem dinheiro para viajar).

Feito para minimizar erros e otimizar tempo

INOVAÇÕES
na programação

CONTROLE PID COM IMU

Na temporada passada, usávamos o giroscópio apenas como apoio dentro do DriveBase. Neste ano, desenvolvemos nosso próprio controle PID, que lê o IMU a cada ciclo, compara o ângulo atual com o ângulo desejado e corrige a diferença de potência entre os motores em tempo real.
Com isso, o robô deixa de depender da potência bruta dos motores, que varia conforme a bateria descarrega. A correção passa a ser feita sobre o erro de direção, e o resultado se mantém ao longo do dia de competição.

ROTAS COM HEADING ABSOLUTO

Em vez de zerar o giroscópio a cada movimento, zeramos uma única vez, no início da rota. Cada passo passa a mirar um ângulo absoluto de campo, não um ângulo relativo ao ponto onde o robô parou.
Antes, um pequeno erro no fim de um giro virava o novo “zero” e nunca era percebido. Agora esse erro continua visível para o sistema, e o movimento seguinte o corrige. Mesmo iniciando a rota com uma pequena diferença de orientação, o robô converge para a trajetória correta.

DRIVE_BASE.RESET()

```python
class PID:
    def __init__(self, kp, ki, kd, integral_limit=None):
        self.kp, self.ki, self.kd = kp, ki, kd
        self.integral_limit = integral_limit
        self.integral = 0
        self.last_error = 0

    def reset(self):
        self.integral = 0
        self.last_error = 0

    def update(self, error, dt):
        self.integral += error * dt
        if self.integral_limit is not None:
            self.integral = max(-self.integral_limit, min(self.integral_limit, self.integral))
        derivative = (error - self.last_error) / dt if dt > 0 else 0
        self.last_error = error
        return self.kp * error + self.ki * self.integral + self.kd * derivative
```

DESAFIOS
na programação

Ruído do giroscópio

Ao aumentar a força da correção (kp) para o robô reagir mais rápido, percebemos que ele passou a desviar de forma aleatória, sem padrão fixo. O IMU sempre apresenta uma pequena oscilação, e um ganho alto amplificava esse ruído em correções exageradas.
A solução foi reduzir o ganho e adicionar a zona morta. O diagnóstico veio de uma observação importante: desvio sem padrão indica ruído, e desvio sempre para o mesmo lado indica erro de lógica ou de ganho.

Giro com o robô carregado

Nosso método de giro funcionava bem com o robô vazio, mas perdia precisão quando ele carregava peso. Entendemos que o giroscópio corrige erros de direção, mas não compensa falta de força: se o motor não vence a inércia e o atrito, não há “erro de rota” para corrigir.
Passamos a girar em malha fechada, aplicando potência até o heading real atingir o alvo, em vez de pedir o giro uma única vez e confiar que ele termine.





CLASSE PID

Criamos uma classe PID reutilizável, que recebe o erro e o tempo entre ciclos e devolve a correção. Como ela não sabe se vai controlar direção ou giro, o mesmo código serve para os dois.

kp: reage ao tamanho do erro atual

ki: corrige desvios pequenos e persistentes

kd: freia a correção antes de passar do ponto

integral_limit: teto do termo integral (anti-windup)

```python
class PID:
    def __init__(self, kp, ki, kd, integral_limit=None):
        self.kp, self.ki, self.kd = kp, ki, kd
        self.integral_limit = integral_limit
        self.integral = 0
        self.last_error = 0

    def reset(self):
        self.integral = 0
        self.last_error = 0

    def update(self, error, dt):
        self.integral += error * dt
        if self.integral_limit is not None:
            self.integral = max(-self.integral_limit, min(self.integral_limit, self.integral))
        derivative = (error - self.last_error) / dt if dt > 0 else 0
        self.last_error = error
        return self.kp * error + self.ki * self.integral + self.kd * derivative
```

MOVIMENTAÇÃO

A função move_straight percorre uma distância mantendo o robô apontado para um ângulo alvo. A distância é medida pelos encoders, usando o diâmetro real da roda (54 mm), e não por tempo. Por isso o robô percorre a mesma distância mesmo com a bateria mais fraca.

```
def move_straight(hub, left_motor, right_motor, distance_mm, speed):
    heading_pid = PID(kp=2.0, ki=0.0, kd=0.1, integral_limit=20)
    hub.imu.reset_heading(0)

    left_motor.reset_angle(0)
    right_motor.reset_angle(0)
    target_angle = (distance_mm / (math.pi * WHEEL_DIAMETER)) * 360

    timer = StopWatch()
    last_time = timer.time()

    while (abs(left_motor.angle()) + abs(right_motor.angle())) / 2 < target_angle:
        now = timer.time()
        dt = (now - last_time) / 1000
        last_time = now
```

A cada ciclo, o erro é a diferença entre o ângulo alvo e o ângulo lido no IMU

A correção é somada em um motor e subtraída no outro

O giroscópio não é zerado aqui, porque a referência vem da rota

GIRO POR heading

Em vez de pedir “gire X graus” e confiar que o movimento termina, a função turn_to_heading gira até o ângulo real do IMU chegar ao ângulo alvo. Se o robô travar por um instante (peso, atrito), ela continua aplicando potência até completar.

O alvo é um ângulo absoluto de campo, não relativo

A tolerância (2°) evita que o robô fique tremendo tentando acertar o último grau

```
def turn_to_heading(hub, me, md, target_heading, speed=150, tolerance=2):
    while abs(target_heading - hub.imu.heading()) > tolerance:
        error = target_heading - hub.imu.heading()
        direcao = 1 if error > 0 else -1
        me.run(direcao * speed)
        md.run(-direcao * speed)
        wait(10)
    me.brake()
    md.brake()
```

ROTAS

Recursividade de código

Separamos as rotas em um arquivo próprio, onde cada rota é uma função que só descreve a sequência de ações do robô. Elas não sabem como o robô se move: chamam comandos prontos de movimentação (andar, rotate) e de garra (close, openn), que ficam em outros arquivos. Assim, conseguimos ajustar uma missão sem risco de interferir nas outras, e podemos melhorar a movimentação sem reescrever nenhuma rota.

```
def rotaI():
    bipar(500,100)
    andar(18)
    rotate(90)
    wait(500)
    andar(8)
    close()
    wait(500)
    andar(-15)
    wait(500)
    rotate(-90)
```

- Toda rota começa com um sinal sonoro (bipar), que confirma que ela iniciou
- Rota I: anda 18 cm, gira 90° para se alinhar, avança 8 cm até o objeto, fecha a garra, recua 15 cm e gira -90° para voltar à orientação original
- Rota II: anda 14 cm, gira 90°, avança 4 cm e abre a garra para soltar
- Pausas curtas (wait(500)) entre as ações dão tempo para o robô estabilizar antes do próximo movimento

```
def rotaII():
    bipar(500,100)
    andar(14)
    rotate(90)
    andar(4)
```

ANTI-WINDUP
e deadband

Essas duas proteções evitam que o controle reaja de forma exagerada

```
self.integral = max(-self.integral_limit,
                    min(self.integral_limit, self.integral))


if abs(error) < 1.0:
    error = 0
```

O termo integral do PID soma o erro ao longo do tempo para corrigir desvios pequenos e persistentes. O problema é que ele continua somando enquanto o erro existir.

Para evitar isso, colocamos um teto no valor acumulado (integral_limit). Mesmo depois de um bloqueio longo, a contribuição do termo integral nunca passa de um máximo conhecido, e a correção volta ao normal assim que o robô se solta.
Sem limite: o erro acumulado cresce enquanto o robô está preso e causa uma reação exagerada ao ser liberado
Com limite: o acúmulo trava no teto, e o robô retoma a trajetória de forma controlada
O limite é aplicado a cada ciclo, logo depois da soma

```python
def update(self, error, dt):
    self.integral += error * dt

    # anti-windup: trava o acúmulo no teto
    if self.integral_limit is not None:
        self.integral = max(-self.integral_limit,
                            min(self.integral_limit, self.integral))

    derivative = (error - self.last_error) / dt if dt > 0 else 0
    self.last_error = error
    return self.kp * error + self.ki * self.integral + self.kd * derivative
```

DEADBAND

O giroscópio nunca lê um ângulo perfeitamente estável: mesmo com o robô parado, o valor oscila um pouco. Em um controle com ganho baixo, esse ruído passa despercebido. Mas, quando aumentamos o ganho para o robô reagir mais rápido, o PID passou a tratar essa oscilação como desvio real e a corrigir “fantasmas”. O resultado foi um robô que desviava de forma aleatória, sem padrão fixo.

A solução foi uma zona morta (deadband): se o erro de ângulo for menor que 1°, o controle o considera zero e não corrige. Só desvios maiores que o ruído do sensor geram resposta.

```python
DEADBAND = 1.0

error = target_heading - hub.imu.heading()
if abs(error) < DEADBAND:
    error = 0
correction = heading_pid.update(error, dt)
```

SOFTWARES utilizados

Recursos Humanos

Para que a equipe de engenharia do robô pudesse fazer um trabalho colaborativo mesmo distante do local de treino e desenvolvendo habilidades para o mercado de trabalho e nosso desenvolvimento humano, utilizamos a ferramenta em nuvem OneDrive. Dessa forma, temos um registro contínuo de tudo que fizemos no torneio além do caderno de engenharia que também fica disponível na nossa pasta compartilhada





Divulgação

Como programamos o robô em blocos, utilizamos o **GitHub** para armazenar e compartilhar a programação na nuvem, mantendo o histórico de alterações. O link do repositório está disponível em nosso Instagram.
Para divulgar o design do robô, utilizamos o **BrickLink Studio 2.0**, que permite modelar e disponibilizar o projeto para visualização online e download.









CONTATOS

Nossos meios de comunicação!

(21) 98518-6299

alphatbrrj@gmail.com

@alphatbr

Firjan SENAI SESI Jacarepaguá