Benchmark Models on Their Ability to Navigate AIM Robot

Meher Kaur Brar and Abhay Yadav · Nov 11, 2026

Woodland School, Portola Valley, CA, USA

DOI: 10.5281/zenodo.24387747

Abstract

The study investigates the feasibility of using Large Language Models (LLMs) to write code that can navigate an AIM robot through a series of challenges. Once AI-generated code could successfully control the robot to autonomously perform a series of tasks with varying difficulties, we then tested the same tasks using different frontier lab models to generate the code and then created a benchmark to test how each of these models met the objective of the test to perform the series of tasks and how they compare across a variety of metrics.

The experiments demonstrated the differing capabilities between models: Muse Spark 1.3, Claude Opus 5.5, and GPT-6 Astra. Through our trials we noted that Opus performed better overall, however both Muse and Astra were less equipped to deal with this task.

1. Introduction

1.1 Motivation

Our motivation on this matter is that we are two eighth graders who are interested in robotics who were wondering about the feasibility of using AI models for coding robots. The broad version of the problem we are studying is: can AI do a robotics challenge in real life with a robot with real life problems like physics?

It matters because it's important to see if AI can solve problems in the real world where things can go wrong much more easily. It also matters because in the future AI is going to have to solve problems in the real world, an example is the AI-robot-arm-smoothie-makers.

1.2 From Robotics to the Research Question

We have been doing robotics for many years. Our team started in 2022 when we started doing VEX IQ robotics. Since then AI has become very prominent in the world, and after noticing some April tags in the new game field for Mix and Match, we wondered if it would be possible to have AI code our robot. For this our robot would need to be very precisely built, which is quite difficult.

Then we found out about VEX AIM, which consists of a mini robot with a kiwi drive, some April tag targets, magnetic barrels, magnetic soccer balls, and an 18'' by 18'' field. We tested if models could code the mini robot to knock down the April tag targets with the magnetic soccer ball, and it turns out they could. Next we made a harness which has all the models in one place and can directly access the robot, that way we could start our investigation.

1.3 Research Questions

- **Q1:** How do different LLMs differ when given a task to solve?
- **Q2:** How can different LLMs cope with a situation in the real world?
- **Q3:** Can an LLM code a script to run a robot using only images and the code library?
    - **Q3.5:** How do the different scripts differ?

1.4 Hypotheses

- **H1:** Models will demonstrate significantly different performance.
- **H2:** Performance will decrease as task difficulty increases.
- **H3:** There will be more errors when there are more objects on the field.
- **H4:** The performance will decrease with fewer images given.

2. Related Work

During the process of creating this paper we reviewed a few research papers on similar topics. The two main papers [1, 2] highlighted the necessity of a key scoring rubric that had to be defined. Existing benchmarks mostly show the rubrics and designs for the benchmarking but do not actually benchmark any of the LLMs. Others that do test the code solely test the code in a simulated environment.

The important question that this paper answers is what would actually happen if we were to test the LLMs at tasks in the physical world.

3. Benchmark Design

3.1 Benchmark Prototype

Initially we tried to see if having the robot code for the VEX AIM robot was even possible. We did a series of experiments to prototype. When prototyping we did everything more manually.

We started by creating a game with the ball and the purple April tag targets. We laid out the parts depending on how difficult we wanted the game to be. This allowed us to see the limits of what we could do. Each game ChatGPT and Claude.ai would get three attempts. Then we took a picture of the game with a ruler and sent that to ChatGPT or Claude.ai along with our prompt. Our prompt consisted of the rules of the game, what attempt number it was on, and the starting picture.

Initially we tried using Meta's Muse however it was unable to complete the first and simplest task. It also took a long amount of time to generate, hence the exclusion from all further prototype experiments.

We then had ChatGPT or Claude.ai generate code based on what it felt was appropriate. Once the code generated we would copy and paste it into the VEXcode editor for AIM. Once the code was copied in we would upload it to the robot and then run the code. While the robot was running we would be taking photos at 1 second intervals or more. After the robot completed or failed the game we would pass 10 interesting images to ChatGPT and Claude.ai alongside the attempt number and how many targets it knocked down. ChatGPT and Claude.ai would then generate the second or third attempt's code. Once generated we would once again copy the code into a new file on the VEXcode editor and run it.

During the second attempt we would continue taking photos during the run. After both ChatGPT and Claude.ai completed their second attempt we would take the images of that attempt and paste them into the conversation with either ChatGPT or Claude.ai. After that we would state the attempt number and tell either ChatGPT or Claude.ai to generate the third attempt code. We ran this test for four games before concluding that ChatGPT and Claude.ai can create code for a VEX AIM robot. In the fourth game we introduced the concept of after the first and second run we would provide the console logs from the robot to ChatGPT and Claude.ai.

In all, one attempt could take up to eleven minutes including the time ChatGPT or Claude.ai took to think, the time it took to get everything ready, and the time for the run. ChatGPT or Claude.ai took about three to five minutes to think, it took around a minute to

get everything ready including copy and pasting code and running the robot, and it took anywhere from 1 second to 5 minutes to run.

3.1.1 Insights and Learnings

Some mistakes that could have led to inconsistencies in the data during prototyping that could have occurred were human errors, models' websearch tool, and human judgement. Some human errors that might have occurred were inaccurate picture taking. For example taking a picture every two seconds versus accidentally taking a picture every one second. We didn't know if we were talking to the model or their websearch tool so they might be interpreting the prompts differently, and we would be benchmarking their websearch tool instead of the model.

We also used our own judgement a few times. For example, while prototyping, we would give seven to ten pictures to the model and would choose which ones that we thought would be helpful to the model. We would also give each code the model gave one try unless there was an outside problem like the field being tilted so the camera couldn't see clearly. Also if the model completed the game in the second attempt, we wouldn't have a third attempt. These last two rules stayed with our actual test as well.

3.2 Harness Design

To fix these problems and to make runs more efficient we made a harness.

We had Claude make a harness that connects the models, camera, and robot all in one place. There are games where you can choose where to put each barrel, April tag, soccer ball, and robot. In each game there are experiments where you can choose which model you would like to test. In each experiment there are attempts: we would stop the model after 3 attempts or after it succeeded.

We decided models should have multiple attempts because it was difficult to complete the challenge in one attempt and we wanted to see if it could revise its code and improve. In each attempt you can see the prompt, code, telemetry, reasoning, trace, and harness. There would often be problems with connecting the camera to the harness, which needed us to restart the harness in the beginning of each session of collecting data.

The harness would first tell our prompt to the model and have it create a code to complete the challenge. Then it would send the code to the robot over a Wi-Fi connection. While the code is running it would take pictures and have telemetry.

Harness code: github.com/Meher-ilovepandas/robo-bench



Figure 1: Physical setup of the harness

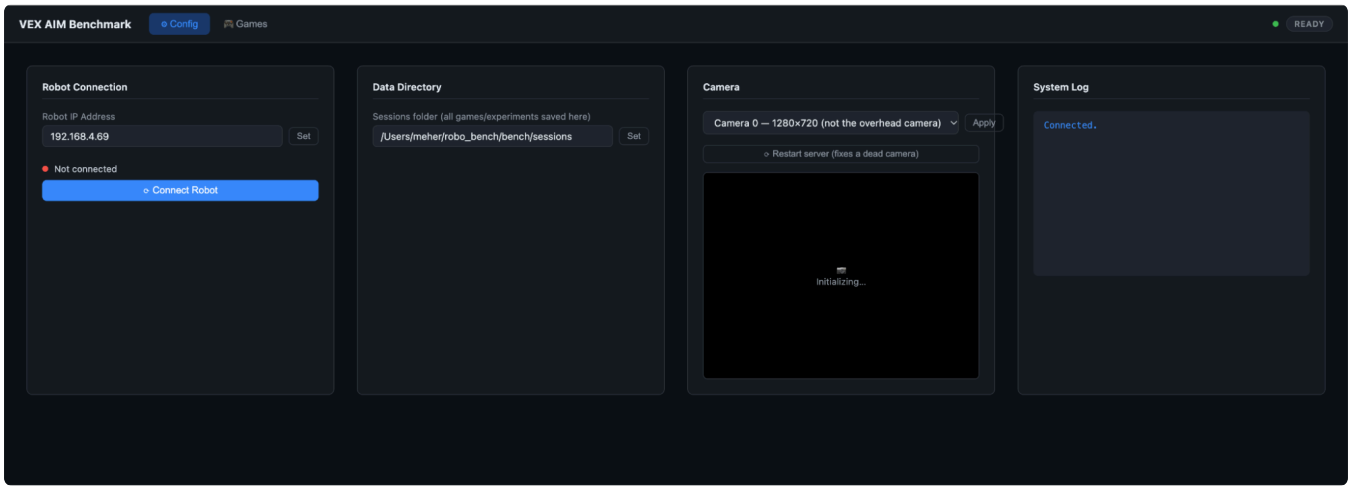


Figure 2: Screenshot of the harness webapp

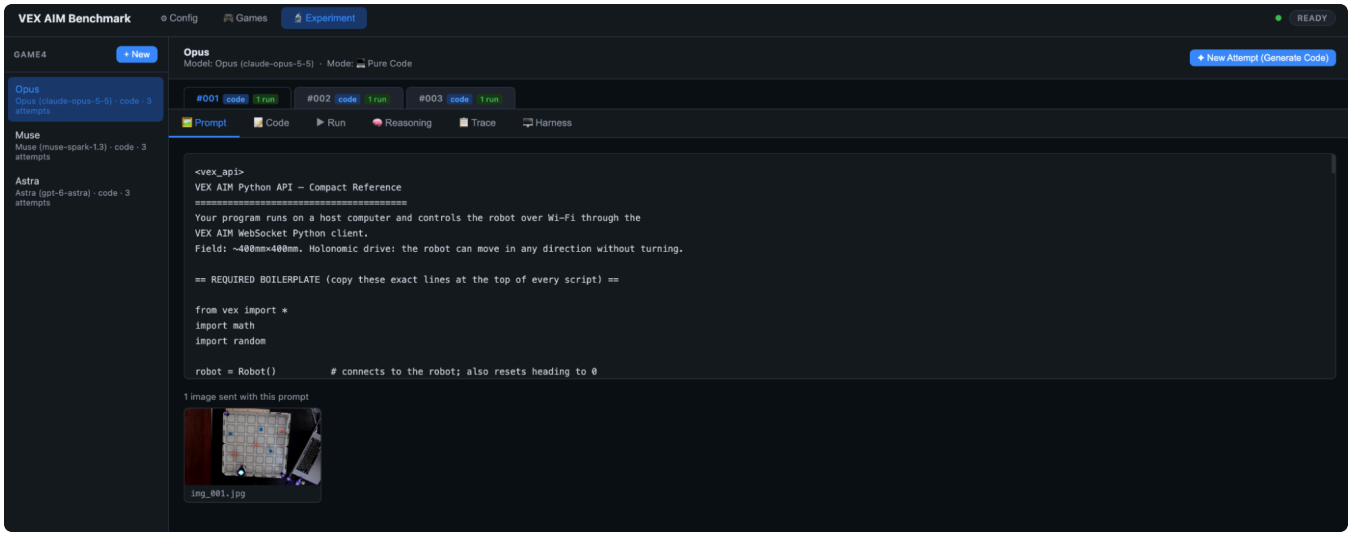


Figure 3: Screenshot of the harness webapp

3.2.1 Harness Prompt Template

The initial prompt that Harness sends to the Model is below. The prompt contains the boilerplate, the entire VEX API library reference, game prompt, starting image of the game.

```
<vex_api>
VEX AIM Python API — Compact Reference
=======================================
Your program runs on a host computer and controls the robot over Wi-Fi
through the
VEX AIM WebSocket Python client.
Field: ~400mm×400mm. Holonomic drive: the robot can move in any direction
```

```
without turning.
== REQUIRED BOILERPLATE (copy these exact lines at the top of every script)
==
from vex import *
import math
import random
robot = Robot()          # connects to the robot; also resets heading to 0
timer = Timer()
robot.vision.model_detection(True)   # AI objects (balls, barrels, robots)
robot.vision.tag_detection(True)     # AprilTags
robot.inertial.reset_heading()
robot.inertial.reset_rotation()
robot.set_xy_position(0, 0)          # odometry is NOT reset automatically
between programs
== MOTION ==
Angles for move_*: 0=forward, 90=right, 180=backward, 270 (or -90)=left,
relative to
the direction the robot is currently facing. Range -360 to 360.
Velocity: PERCENT (default units) 0-100, or MMPS 0-200. 100% move = 200 mm/s.
Default move velocity = 100 mm/s (50%); default turn velocity = 75 deg/s
(~42%).
100% turn = 180 deg/s (units DPS also accepted, max 180).
robot.move_at(angle, velocity=None, units=PERCENT)
. . . (Rest of API Truncated for brevity)
== KICKER ==
. . . (Rest of API Truncated for brevity)
== AI VISION CAMERA (robot.vision) ==
. . . (Rest of API Truncated for brevity)
== TIMING & THREADS ==
. . . (Rest of API Truncated for brevity)
== LOGGING — REQUIRED ==
. . . (Rest of API Truncated for brevity)
</vex_api>
<game_description>
(Game specific prompt goes here)
</game_description>
<first_image>
Overhead camera view of the field at the start of the game:
[image: img_001.jpg]
</first_image>
Write the complete Python program for attempt 1.
```

Subsequent prompts used by harness to ask the model to repair its code.

```
<vex_api>
VEX AIM Python API — Compact Reference
======================================
Your program runs on a host computer and controls the robot over Wi-Fi
through the
VEX AIM WebSocket Python client.
Field: ~400mm×400mm. Holonomic drive: the robot can move in any direction
without turning.
== REQUIRED BOILERPLATE (copy these exact lines at the top of every script)
==
from vex import *
import math
import random
robot = Robot()          # connects to the robot; also resets heading to 0
timer = Timer()
robot.vision.model_detection(True)   # AI objects (balls, barrels, robots)
robot.vision.tag_detection(True)     # AprilTags
robot.inertial.reset_heading()
robot.inertial.reset_rotation()
robot.set_xy_position(0, 0)          # odometry is NOT reset automatically
between programs
== MOTION ==
( . . . rest of API library truncated for brevity )
</vex_api>
<game_description>
(per game description prompt)
</game_description>
<first_image>
Overhead camera view of the field at the start of the game:
[image: img_001.jpg]
</first_image>
<previous_code attempt="1">
(Entire code generated in previous attempt)
</previous_code>
<previous_console_logs attempt="1">
(Full console logs generated by the model’s own code)
</previous_console_logs>
<previous_telemetry attempt="1">
(Telemetry generated by Vex AIM robot for the duration of the run)
[
Example:
t,heading,pitch,roll,yaw,robot_x,robot_y,battery,accel_x,accel_y,accel_z,gyro
_x,gyro_z,ai_objects,ai_type,ai_x,ai_y,ai_score
```

```
0.26,0.006,-0.933,-0.456,0.006,0.000,0.000,72,-0.017,0.009,-0.984,0.000,0.000
,2,aiobj,248,57,64
. . .
]
</previous_telemetry>
<previous_images attempt="1">
Overhead camera frames from each trial, one every 2s, in chronological order.
Trial 1, t≈0s:
[image: img_002.jpg]
Trial 1, t≈2s:
. . .
Trial 1, t≈54s:
[image: img_029.jpg]
</previous_images>
Above are the code and results of attempt 1 (1 trial). Write the complete
Python program for attempt 2.
```

The full prompts and Model responses for every run are available at robocodebench.org.

4. Experimental Setup

In a room we set up the 18″ by 18″ VEX field on a table. Above the table we placed a USB web cam that would connect to the computer and harness. Once that was done we created the game accordingly and ran the game.

During the time the model was running one of us would be writing notes of how it performed and what its behavioural patterns were like. The other person would be monitoring the outputs from the robot on the harness. After the run was done we would initiate another trial. If the robot passed we would mark it as complete and move to the next model.

While the model was generating the next attempt code we would reset the field while looking at the reference picture sent to the model as the prompt. We would give the model a max of three attempts. Once we completed the attempts of one model we would move onto the next and so on for each game.

4.1 Models Evaluated

We evaluated Muse Spark 1.3, GPT-6 Astra, and Claude Opus 5.5. We found that the models seem to have personalities. Muse isn't the smartest and often goes in circles, but it's very determined. Astra is more on the cautious side by moving in small increments.

Opus is very confident by moving directly and aiming accurately from far away to save time.

4.2 Scoring Rubric

We would score the robot on three different metrics.

First we would see how many attempts the robot took to complete the game. This would be important as it would show efficiency, learning skill, and skill.

Secondly we would track the amount of time the robot took. This would be used to compare the efficiency of the models.

Finally we would measure the amount of time the model took to generate the code and the complexity of the code. This would be done by timing the generation time and by checking for length of code. The complexity of the code was not a primary metric in this benchmarking but was interesting to note.

5. Games

We created 4 games in increasing order of difficulty. This allowed us to measure how each model performs at varying difficulty levels and separate the more capable models from less capable.

5.1 What was measured for each game run

We measure the console logs as generated by the code the model wrote (we strongly encourage the model to write log lines to help it debug later). We capture the telemetry exposed by the AIM robot. We capture frames of the game every 2 seconds.

5.2 Game 1

Game setup. In this game the robot starts from a fixed position touching the field wall. The ball is placed directly in front of the robot in clear line of sight. The target is placed at the center of the field. The game objective is to knock down the target with the ball.

Game difficulty. This is a relatively easy game because the robot has to move a fixed distance forward, which will capture the ball, then it turns 90 degrees left and hits the ball which will knock down the target. We feel this is a fairly easy game that an elementary/middle school level student can write.

Game prompt.

```
You are going to autonomously drive (by writing python code) a VEX AIM robot
We have given you details about the library of functions to control and
program the VEX AIM robot.
See the library, read it carefully to understand the different commands and
instructions and their behavior.

The goal of the game is to knock down the purple, April tag and the robot
will be in the same place at the start of each turn. Once you have done that
you have completed the game.
I added a ruler to the field just to show you dimensions.
A VEX AIM Field uses individual tiles that measure 9" x 9" each, coming in a
standard kit that includes 4 field tiles and 8 wall pieces

Your goal is to write a VEX AIM python program that plays the game described
below.
Using the picture of the game board provided
Game objective:
Grab the yellow ball by moving close to it (its magnetic so it will roll and
attach if you get close enough). and then hit the purple target (Apri Tag)
using the ball. Do not ram into the tag.

A few things to note:
1. Ball is gently held by magnet to the robot, if you get near the wall and
rotate with the ball attached, it will get dislodged and you will lose the
ball. So do not take the ball and go near the wall and start rotating.
2. The robot has camera sensor which will show you the target april tags, it
has lot of telemetry also. Keep in mind the april tags are flat (very thin
width), if you look from front you will see clearly, if you look from side
you will not see them.
3. In the picture its taken from above, the april tag will appear as a small
purple circle with a line through it. You won't be able to see the april tag,
only robot camera can spot it
```

Scoring criteria. Scoring criteria is for the robot to knock down one target using the ball in 60 seconds. It's a boolean score success/fail.

Start. Starting image sent as part of the prompt to the model, shows the game setup taken by a fixed USB camera from top down.

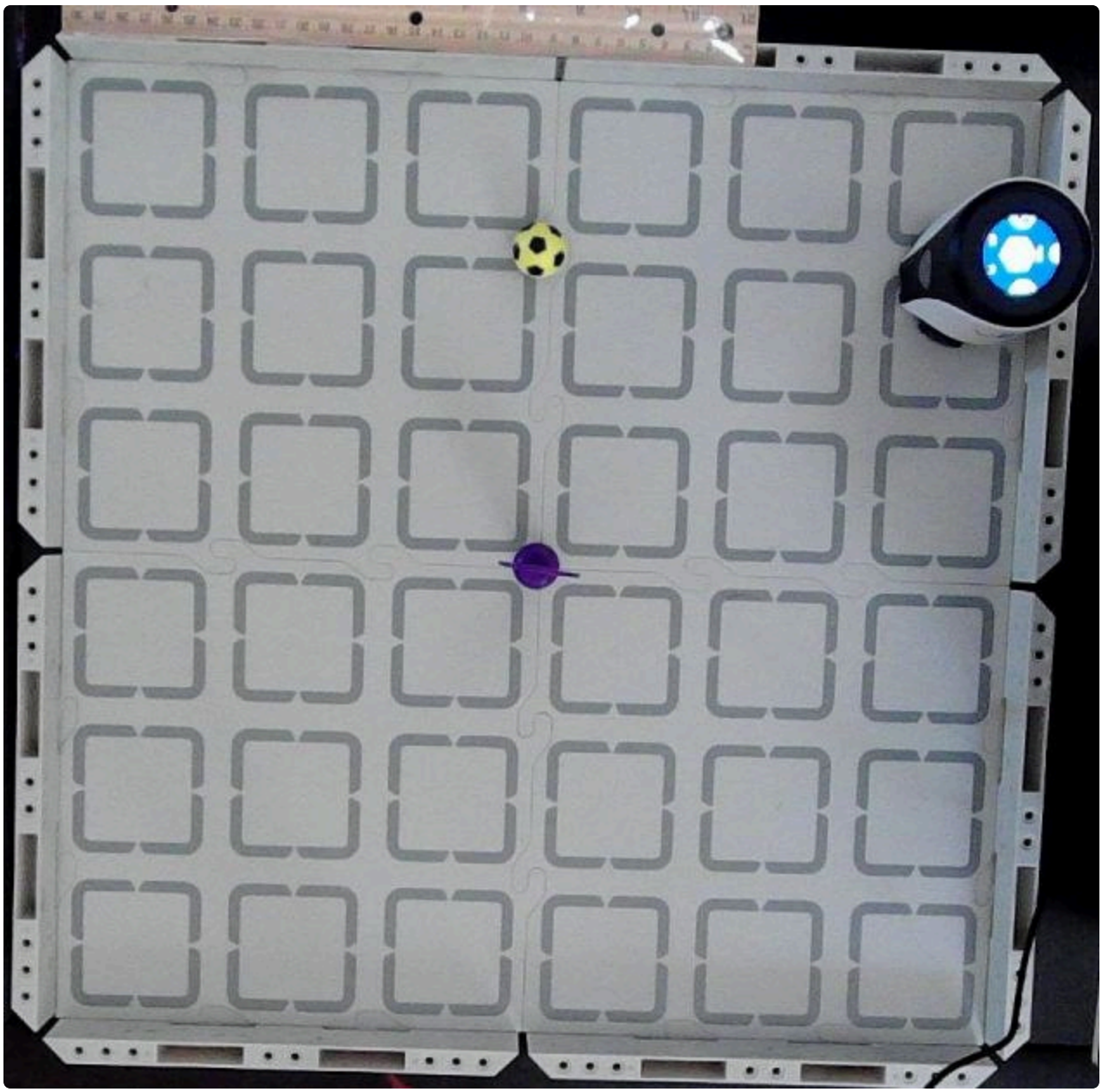


Figure 4: Game 1 start

Success example. Below is an image from one of the successful runs by Opus 5.5. The robot successfully launched the ball and hit the target knocking it down.

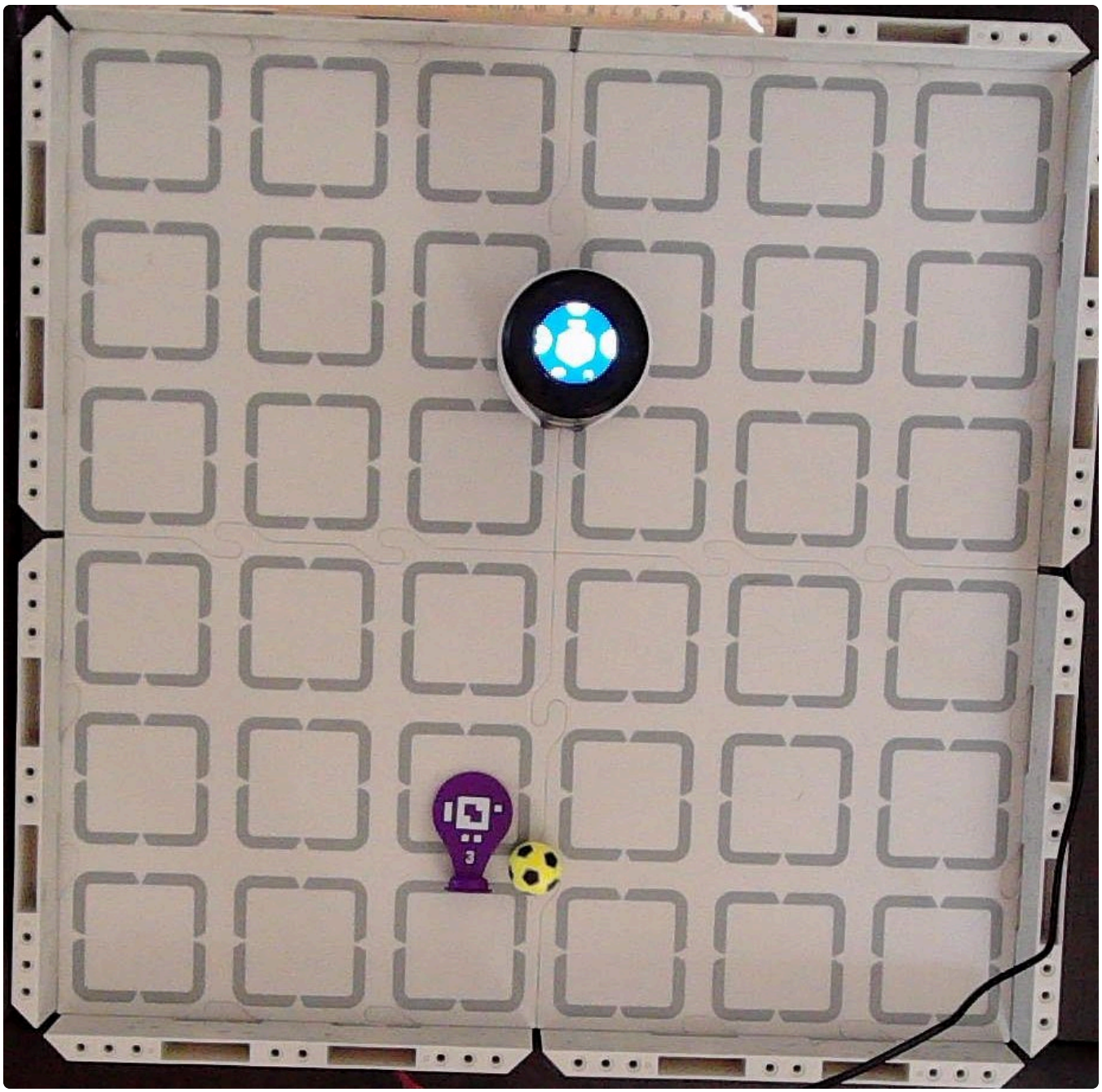


Figure 5: Game 1 success example

Fail example. This is a sample final image of a failed attempt by Muse. The robot got stuck in the corner holding the ball, while the target remains untouched.

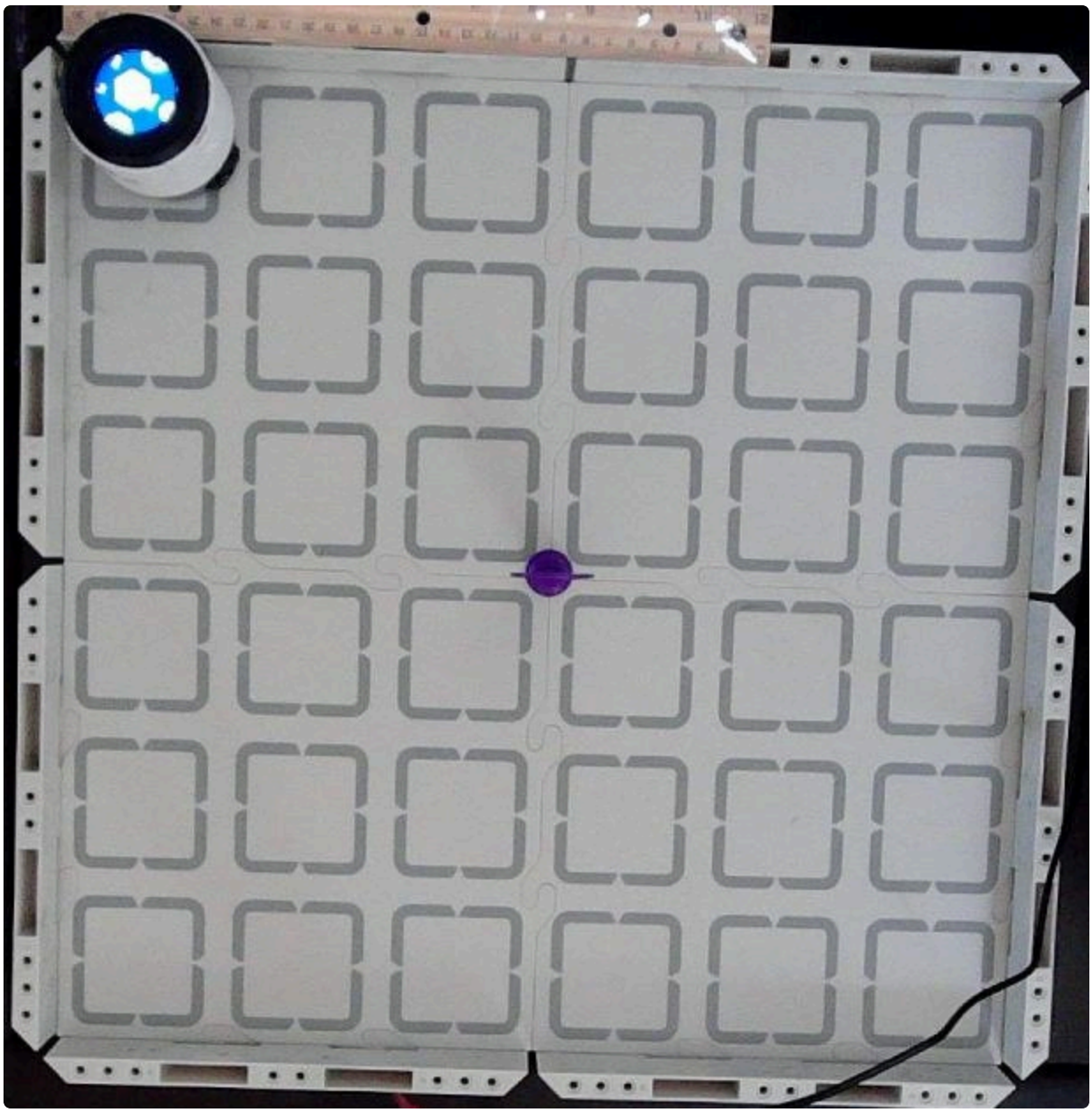


Figure 6: Game 1 fail example

5.3 Game 2

Game setup. In this game the robot started from the wall but not facing the ball. There were 2 targets, one near the wall and the other in a corner. The object of the game was to use the ball to hit both targets.

Game difficulty. This is a bit harder than the first game as the ball isn't right in front of it and it has to go through the act of retrieving the ball so that it can hit the second target. We feel this is a slightly harder game that a middle school/high school level student can write.

Game prompt.

```
You are going to autonomously drive (by writing python code) a VEX AIM robot
We have given you details about the library of functions to control and
program the VEX AIM robot.
See the library, read it carefully to understand the different commands and
instructions and their behavior.

The goal of the game is to knock down the 2 purple, April tag and the robot
will be in the same place at the start of each turn. Once you have done that
you have completed the game.
I added a ruler to the field just to show you dimensions.
A VEX AIM Field uses individual tiles that measure 9" x 9" each, coming in a
standard kit that includes 4 field tiles and 8 wall pieces

Your goal is to write a VEX AIM python program that plays the game described
below.
Using the picture of the game board provided
Game objective:
Grab the yellow ball by moving close to it (its magnetic so it will roll and
attach if you get close enough). and then hit the purple targets (Apri Tags)
using the ball. Do not ram into the tags.

A few things to note:
1. Ball is gently held by magnet to the robot, if you get near the wall and
rotate with the ball attached, it will get dislodged and you will lose the
ball. So do not take the ball and go near the wall and start rotating.
2. The robot has camera sensor which will show you the target april tags, it
has lot of telemetry also. Keep in mind the april tags are flat (very thin
width), if you look from front you will see clearly, if you look from side
you will not see them.
3. In the picture its taken from above, the april tags will appear as a small
purple circle with a line through it. You won't be able to see the april
tags, only robot camera can spot it
4. You will have to go grab the ball again after hitting the first tag that
way you can hit the next one.
```

Scoring criteria. Scoring criteria is for the robot to knock down both targets using the ball in 60 seconds. It's a boolean score success/fail.

Start. Starting image sent as part of the prompt to the model, shows the game setup taken by a fixed USB camera from top down.

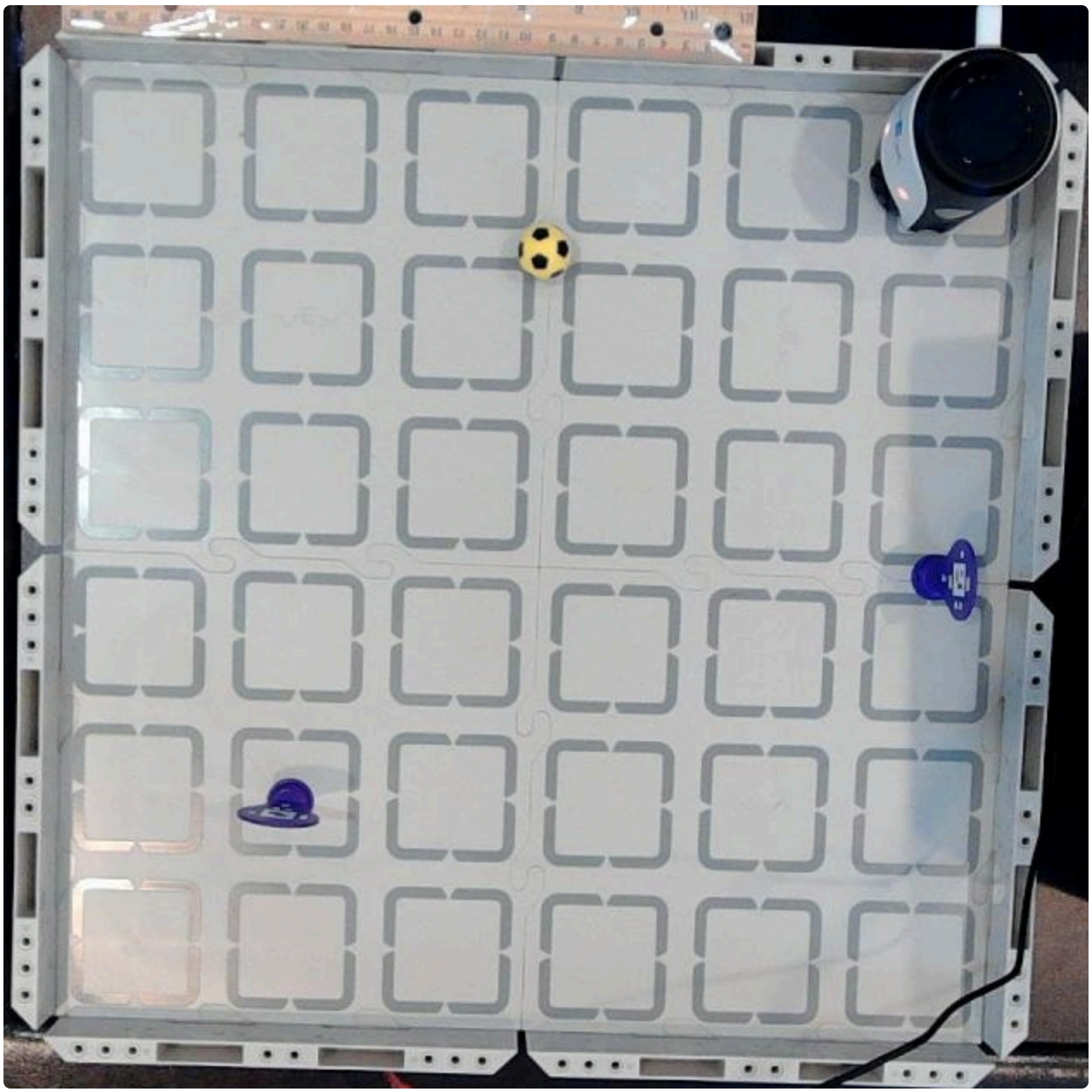


Figure 7: Game 2 start

Success example. Below is an image from one of the successful runs by Muse Spark 1.3. The robot successfully hit both targets and retrieved the ball.

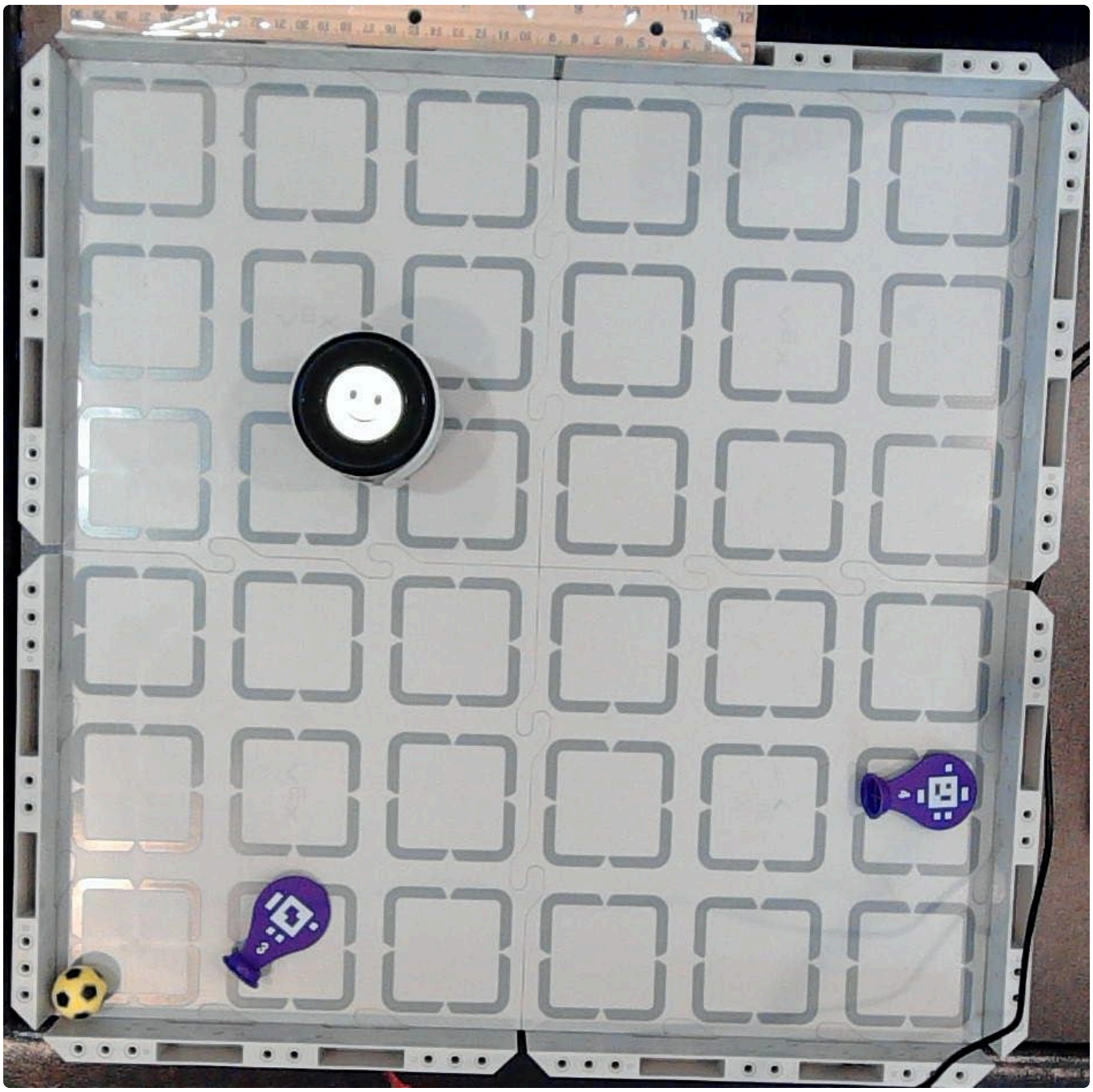


Figure 8: Game 2 success example

Fail example. This is a sample final image of a failed attempt by Astra. The robot couldn't find the ball, and the second target remained untouched. It spent most of its time spinning in circles.

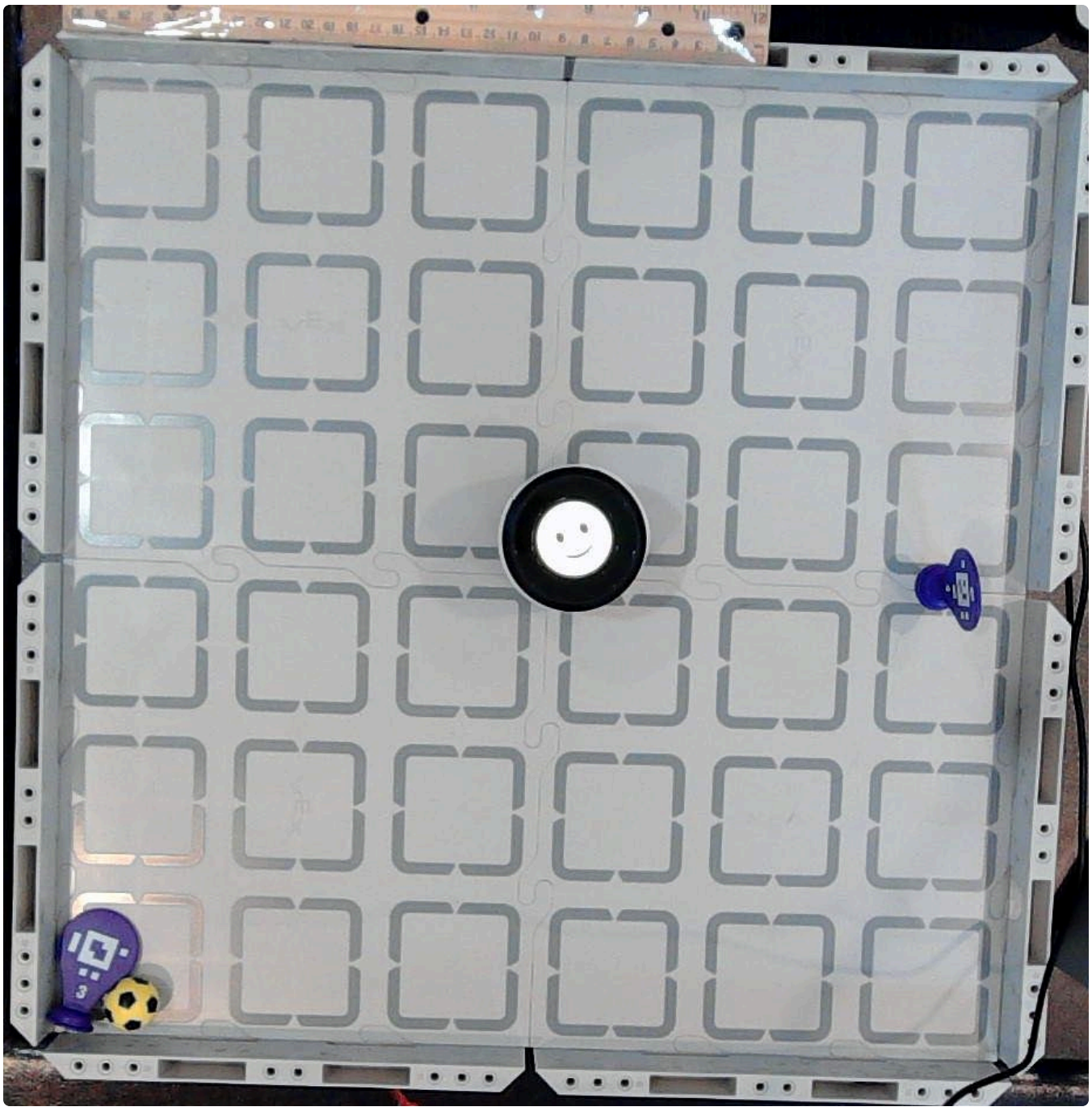


Figure 9: Game 2 fail example

5.4 Game 3

Game setup. In this game the robot started from the wall facing the ball. There were 2 targets, one near the wall and the other in a corner. There were also 2 barrels next to the targets. The object of the game was to use the ball to hit both targets without hitting the barrels.

Game difficulty. This is a bit harder than the second game as there is another variable with the barrels and getting the right angle is challenging. We feel this is a harder game that a middle school/high school level student can write.

Game prompt.

```
You are going to autonomously drive (by writing python code) a VEX AIM robot
We have given you details about the library of functions to control and
program the VEX AIM robot.
See the library, read it carefully to understand the different commands and
instructions and their behavior.

The goal of the game is to knock down the 2 purple, April tag and the robot
will be in the same place at the start of each turn. Once you have done that
you have completed the game.
You can't directly hit a barrel (if it bounces off of the wall it doesn't
count- it also doesn't count if you run into the barrel with the ball). If
you directly hit the barrel, it counts as a fail.
I added a ruler to the field just to show you dimensions.
A VEX AIM Field uses individual tiles that measure 9" x 9" each, coming in a
standard kit that includes 4 field tiles and 8 wall pieces

Your goal is to write a VEX AIM python program that plays the game described
below.
Using the picture of the game board provided
Game objective:
Grab the yellow ball by moving close to it (its magnetic so it will roll and
attach if you get close enough). and then hit the purple targets (April Tags)
using the ball. Do not ram into the tags.

A few things to note:
1. Ball is gently held by magnet to the robot, if you get near the wall and
rotate with the ball attached, it will get dislodged and you will lose the
ball. So do not take the ball and go near the wall and start rotating.
2. The robot has camera sensor which will show you the target april tags, it
has lot of telemetry also. Keep in mind the april tags are flat (very thin
width), if you look from front you will see clearly, if you look from side
you will not see them.
3. In the picture its taken from above, the april tags will appear as a small
purple circle with a line through it. You won't be able to see the april
tags, only robot camera can spot it
```

```
4. You will have to go grab the ball again after hitting the first tag that
way you can hit the next one.
5. You might have to hit the April tag from a different angle to not hit the
barrel
```

Scoring criteria. Scoring criteria is for the robot to knock down both targets using the ball without hitting the barrels in 60 seconds. If it hit a barrel it would count as a fail. It's a boolean score success/fail.

Start. Starting image sent as part of the prompt to the model, shows the game setup taken by a fixed USB camera from top down.

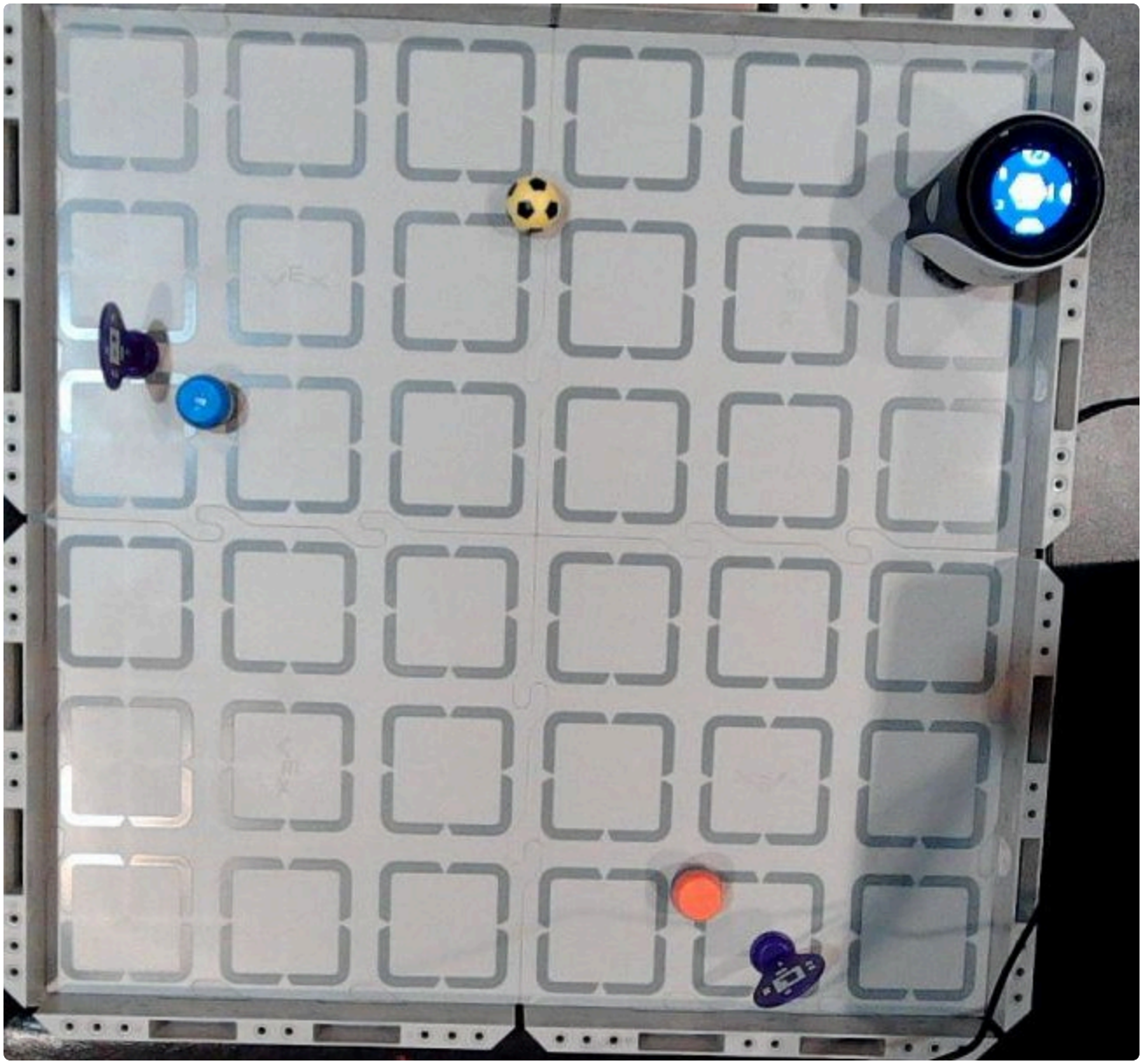


Figure 10: Game 3 start

Success example. Below is an image from one of the successful runs by Opus. The robot successfully hit both targets without hitting the barrels.

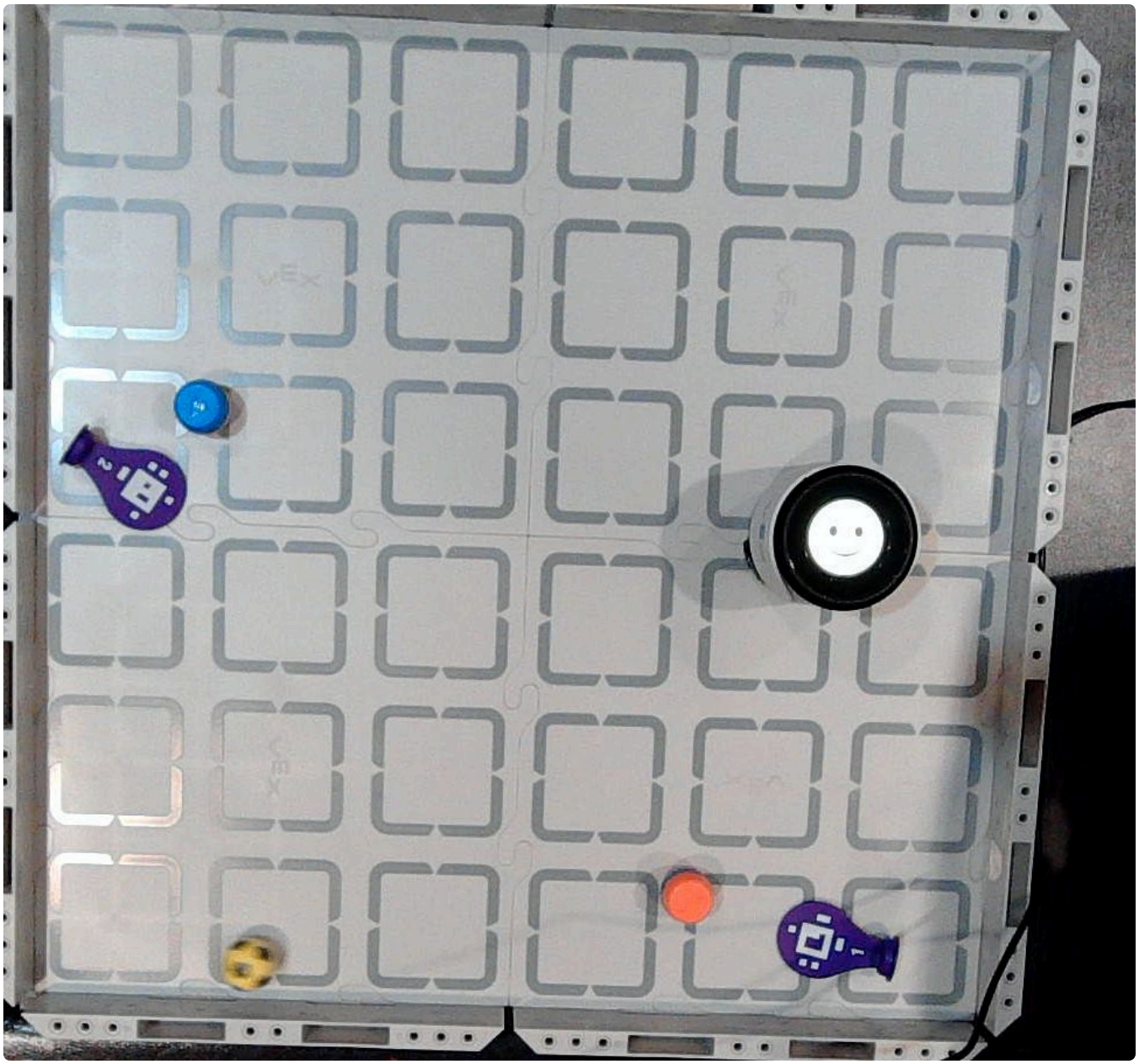


Figure 11: Game 3 success example

Fail example. This is a sample final image of a failed attempt by Muse. The robot couldn't find the right angle to hit the second target with. It spent most of its time pacing back and forth.

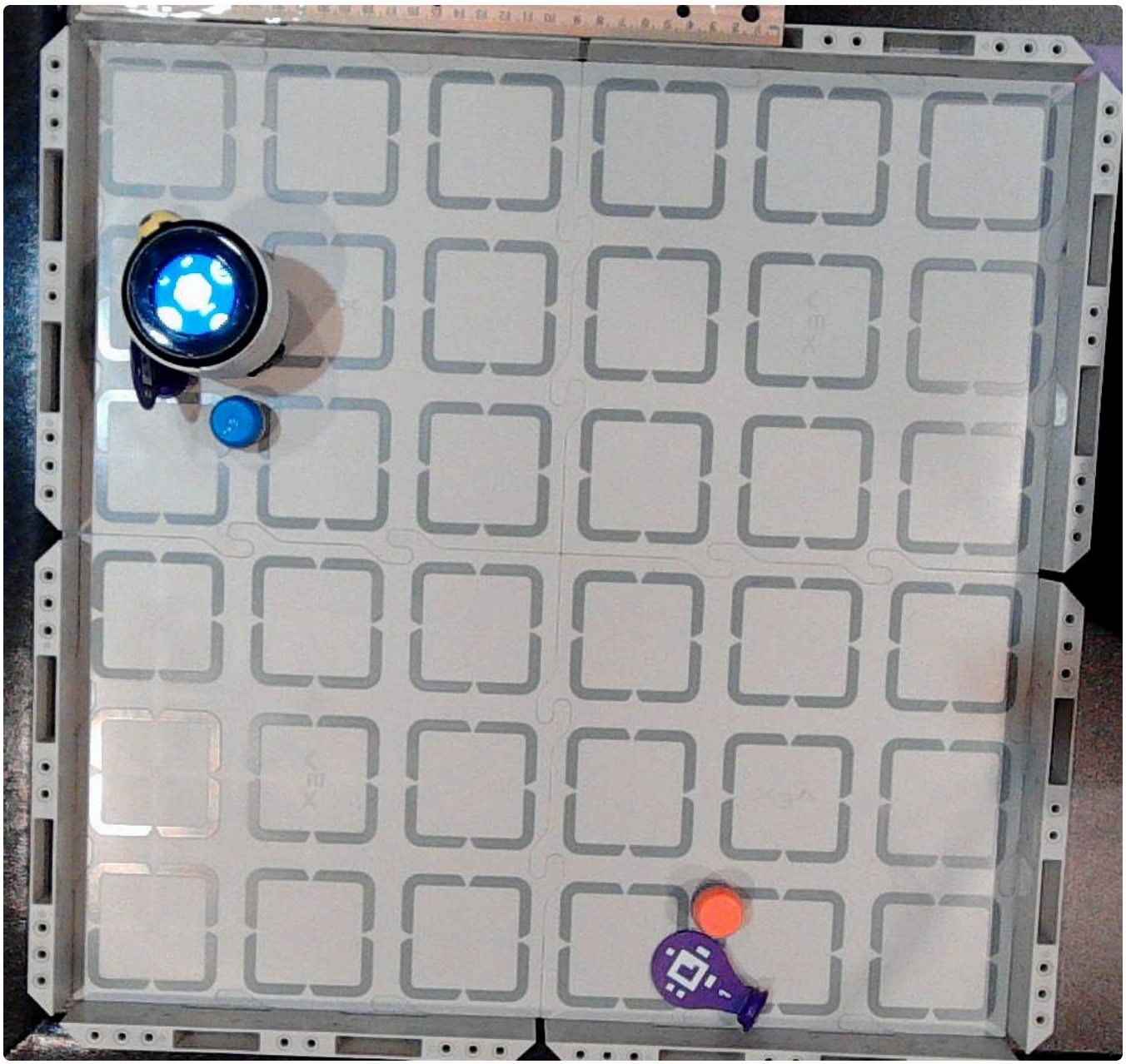


Figure 12: Game 3 fail example

5.5 Game 4

Game setup. In this game the robot started from a different wall facing a lot of barrels. There were 2 targets, one in each corner. There were also 6 barrels in the center that were orange or blue. The object of the game was to sort the barrels by color by putting the orange ones in the bottom corner and the blue ones in the top corner.

Game difficulty. This isn't necessarily harder than the third game but since it was something new and there was little time to complete, the models really struggled. It also appeared that the models misinterpreted one of our rules, but since they all did this we

decided that it was our fault for wording it that way. We feel this is a simple game that an elementary/middle school level student can write.

Game prompt.

```
You are going to autonomously drive (by writing python code) a VEX AIM robot
We have given you details about the library of functions to control and
program the VEX AIM robot.
See the library, read it carefully to understand the different commands and
instructions and their behavior.

The goal of the game is to sort the barrels by color. Place the orange
barrels in the same square the April tag that say 4 is in, and place the blue
barrels in the same square that the April tag that says 0 is in.
I added a ruler to the field just to show you dimensions.
A VEX AIM Field uses individual tiles that measure 9" x 9" each, coming in a
standard kit that includes 4 field tiles and 8 wall pieces

Your goal is to write a VEX AIM python program that plays the game described
below.
Using the picture of the game board provided
Game objective:
Move all six barrels into the correct square within the time limit of 1
minute. You will be graded on the amount of barrels placed correctly and the
time it took.

A few things to note:
1. Barrel is gently held by magnet to the robot, if you get near the wall and
rotate with the barrel attached, it will get dislodged and you will lose the
barrel. So do not take the barrel and go near the wall and start rotating.
2. The robot has camera sensor which will show you the target april tags, it
has lot of telemetry also. Keep in mind the april tags are flat (very thin
width), if you look from front you will see clearly, if you look from side
you will not see them.
3. In the picture its taken from above, the april tags will appear as a small
purple circle with a line through it. You won't be able to see the april
tags, only robot camera can spot it
4. There is a place setting and a kick setting. We would suggest using the
kick setting but you can if you want.
```

Scoring criteria. Scoring criteria is for the robot to sort the barrels by color in 60 seconds. If it got all blue and orange barrels in the correct corners it succeeded. It's a boolean score success/fail.

Start. Starting image sent as part of the prompt to the model, shows the game setup taken by a fixed USB camera from top down.

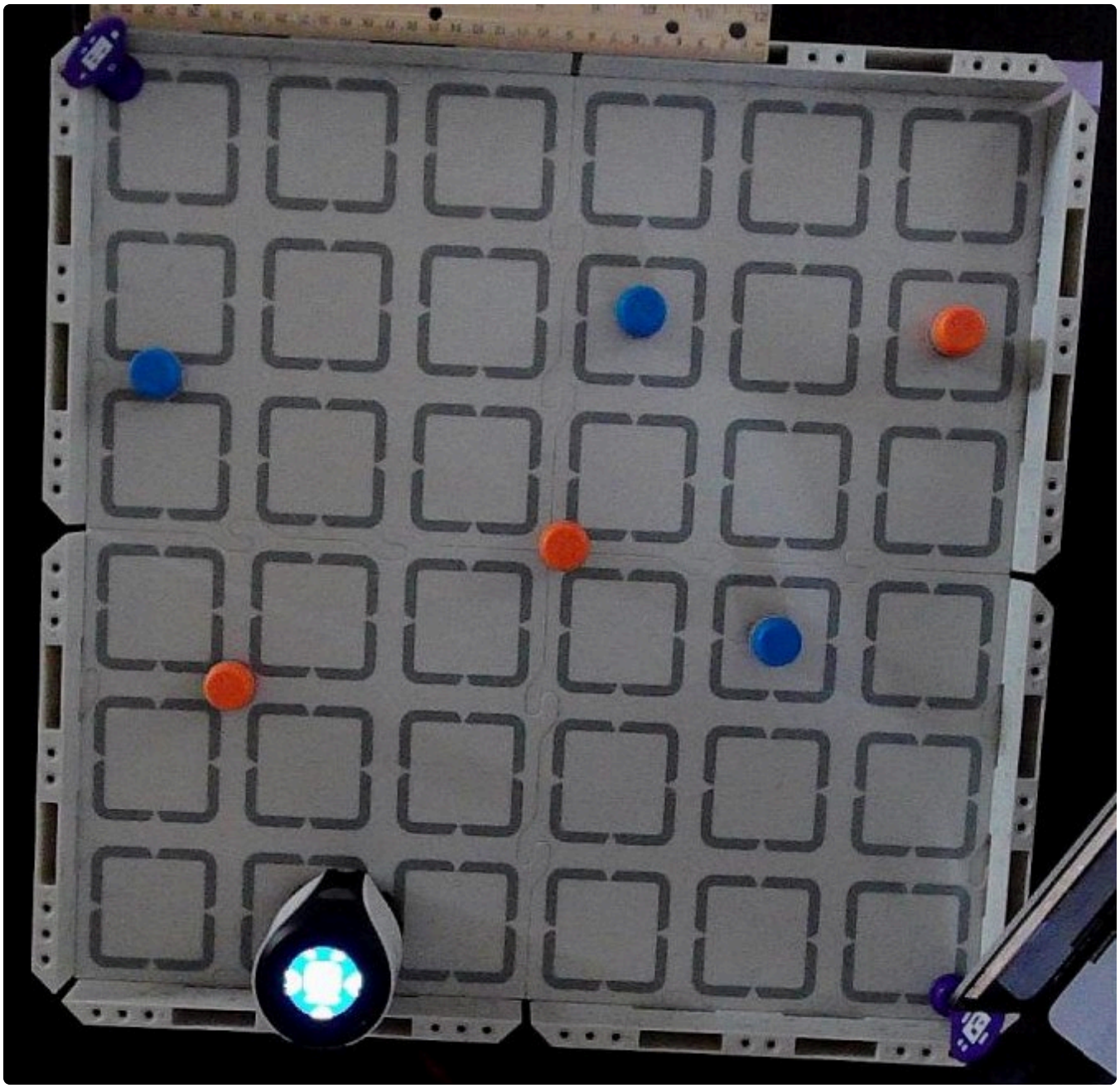


Figure 13: Game 4 start

Success example. No models succeeded in Game 4. Below is an image from the run closest to success by Opus. The robot successfully got ⅔ of the barrels in the correct corners.

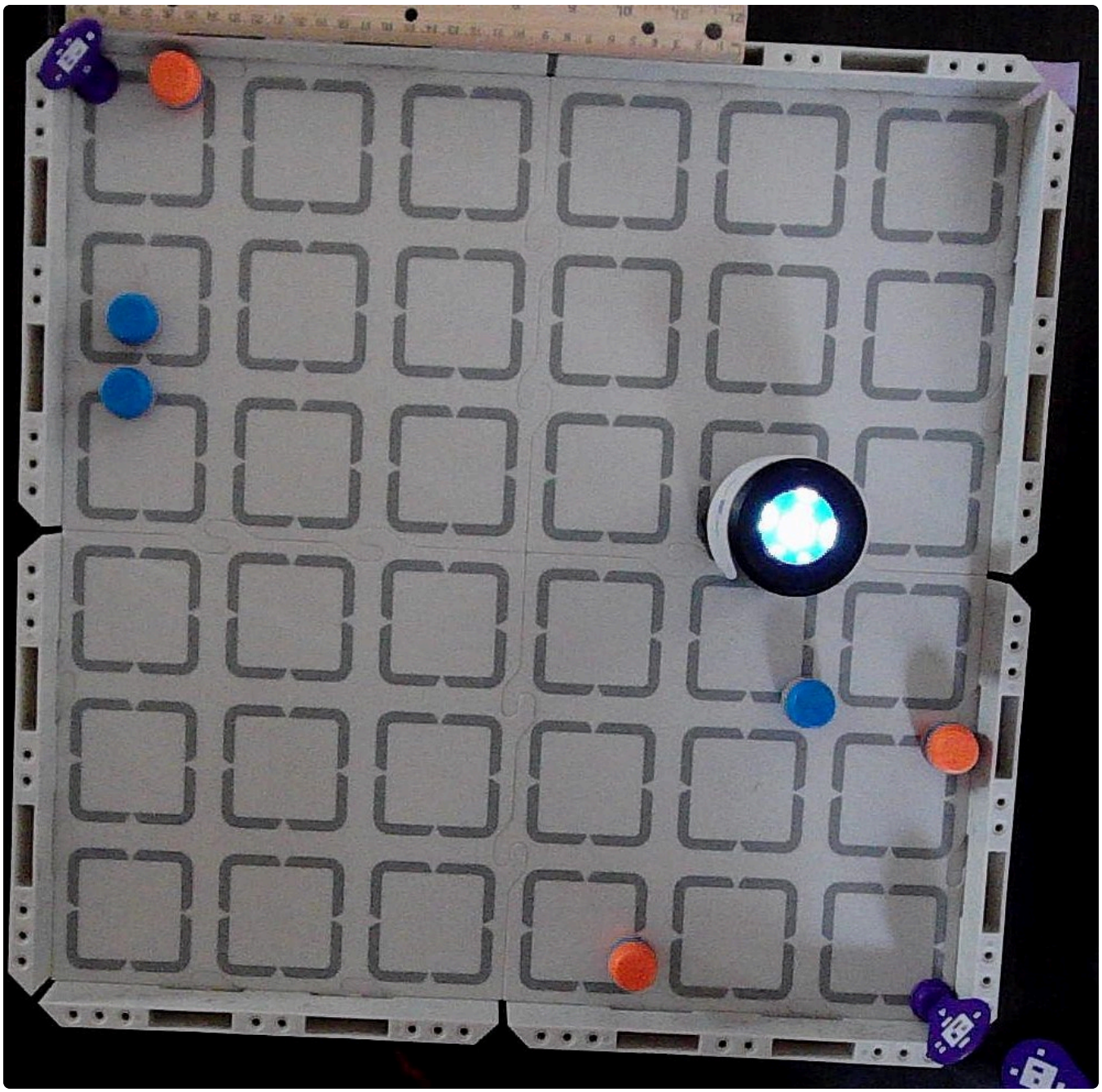


Figure 14: Game 4 success example

Fail example. This is a sample final image of a failed attempt by Muse. The robot couldn't place the barrels in the correct corners.

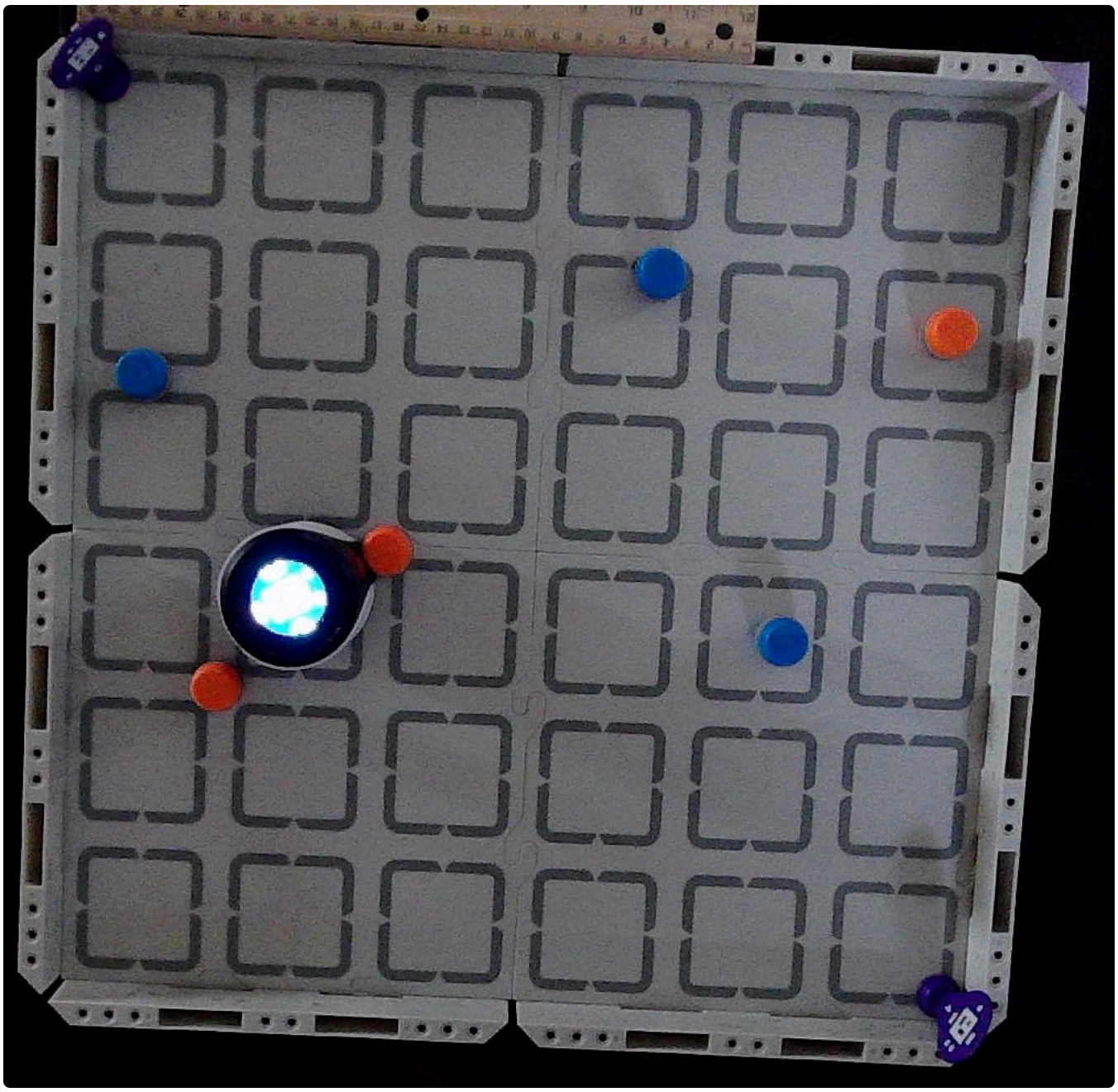


Figure 15: Game 4 fail example

6. Results

We were trying to measure two main things. The first thing we were trying to find was which model was better suited for a task such as controlling a robot in the world. The second thing we were trying to measure was how well each model could revise its code and improve after a failed attempt.

6.1 Chart 1: Model results per game

Opus passed 3 of 4 games, Muse 2, Astra 1

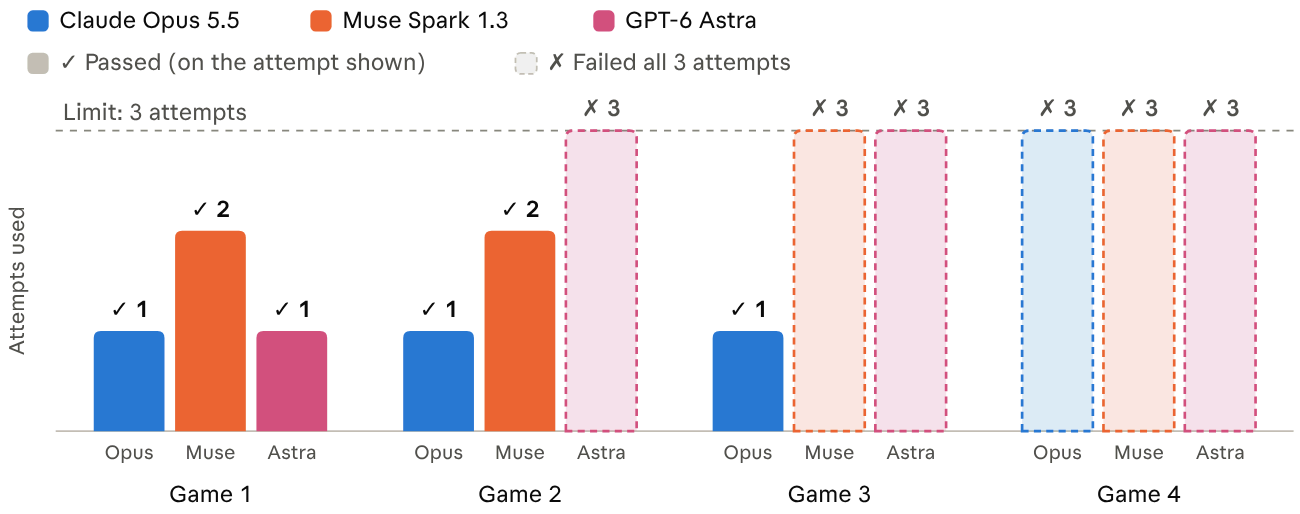
Result on each model's final attempt (up to 3 attempts per game). No model passed Game 4.

	Game 1	Game 2	Game 3	Game 4	Games passed
Claude Opus 5.5	✓ Pass	✓ Pass	✓ Pass	✗ Fail	**3 of 4**
Muse Spark 1.3	✓ Pass	✓ Pass	✗ Fail	✗ Fail	**2 of 4**
GPT-6 Astra	✓ Pass	✗ Fail	✗ Fail	✗ Fail	**1 of 4**

6.2 Chart 2: Model attempts per game

Every game Opus passed took 1 attempt; Muse needed 2

Bar height = attempts used, up to 3. Fewer is better.



6.3 Result Analysis

Opus

Opus worked effectively through quick, accurate decisions. This made its score higher than others. One strength Opus had was to make quick decisions, an example is through how it moves. Unlike other models Opus moved to the ball or target in one accurate stroke instead of in short strokes or realigning as it didn't need to do so.

Another strength Opus had was accuracy which was very helpful as it could shoot the target from far away and didn't have to move closer to hit the target. This helped it save time.

One weakness Opus had was working under the pressure of little time. In the last game when Opus didn't have much time, it panicked by putting the wrong barrels in the wrong corners, just so it could finish putting all the barrels in corners. Opus's personality was accurate and fast, though it valued speed over accuracy. This often helped Opus but occasionally didn't, which can be seen in the fourth game.

Astra

Astra performed decently well but it consistently showed signs of overthinking and overanalysing which caused its results to be lesser than the other models. Astra's strengths included precision, due to its overanalysing it was able to precisely mark each target.

Some weakness Astra had was that due to its overthinking, it would spend a long time trying to find the most optimal angle to hit the tag and would then run out of time. Another weakness is that Astra, when generating the code, would impose stricter time limits separate from the total time, such as a max of 20 seconds to grab the ball. This would often cause failure as the robot would take too much time and time out.

Astra's personality was one of a cautious perfectionist. It would spend a long time trying to find the correct angle and when it did it would slowly inch toward the target to make sure it was correct.

Muse

Muse performed slightly better than Astra however took more turns to succeed on earlier trials. The strengths of Muse were it would adapt quickly after failing one round, it would also be efficient with its movements. Muse seemed to lack spatial awareness and the code was typically bare minimum.

7. Limitations

We only ran each model once on each game, with up to three attempts. Since this is a real robot, things like the battery, the wheels slipping, or the ball rolling a little differently can change what happens in a run. So our results show what happened in our trials, but they don't prove that one model is always better than another.

We planned to also compare how long the robot took and how long each model took to write its code, but we did not include those results in this paper. We also did not test H4, so we can't say if fewer images makes the models do worse.

In Game 4 the prompt told the models they would be graded on how many barrels they placed correctly, but we scored the game as pass or fail. The models also misinterpreted one of our rules, so Game 4 may have been harder than we planned.

Next we want to run each model on each game more times, so we can see how much the results change from run to run.

8. Conclusion

In this paper we tested if AI models can write code that controls a real VEX AIM robot. They can, but they struggle more as the games get harder. In Game 1 all three models passed, in Game 2 two models passed, in Game 3 only Opus passed, and in Game 4 none of the models passed. This matches H2. The models also performed differently from each other, which matches H1, and the games with more objects on the field (Games 3 and 4) had the most failures, which matches H3. Since we only ran each game once, we can't say if these differences are significant. In our trials Opus did the best, passing three of the four games on its first attempt.

Use of AI Tools

We had Claude build the harness code. We also used Claude to organize and format this paper, fix spelling, make the two charts from our results, crop the game photos, fill a few small gaps, and help write the Limitations and Conclusion sections from our results. The experiments, results, and the rest of the writing are our own.

Data Availability

The full benchmark data for every run (prompts, generated code, console logs, telemetry, and camera images) is available at robocodebench.org. The harness code is available at github.com/Meher-ilovepandas/robo-bench.

References

[1] Y. Shen, K. Song, X. Tan, W. Zhang, K. Ren, S. Yuan, W. Lu, D. Li, and Y. Zhuang, "TaskBench: Benchmarking Large Language Models for Task Automation," in *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024. arXiv:2311.18760

[2] A. Reuel, A. Hardy, C. Smith, M. Lamparth, M. Hardy, and M. Kochenderfer, "What Makes a Good AI Benchmark?," Stanford HAI Policy Brief, December 2024. hai.stanford.edu/policy/what-makes-a-good-ai-benchmark