Automatic Antenna Rotor

Jack Kang, Ryan Ma, Rickey McGregor
dongwon900@berkeley.edu, ryan.ma3011@berkeley.edu, rickeymcgregor@berkeley.edu

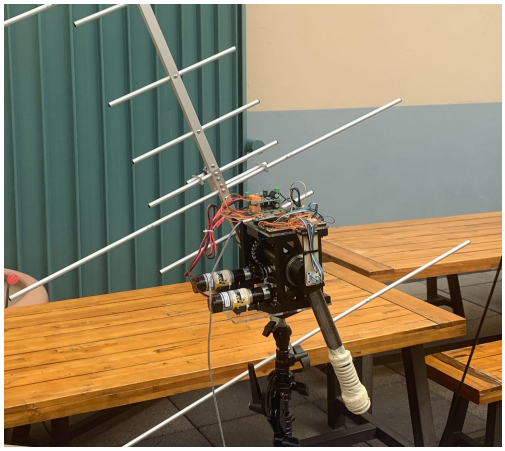
ABSTRACT

Satellite technology has become inseparable from the daily lives of many; it is hard to imagine life without services like GPS, satellite television/internet, satellite image maps, etc. Though this technology has largely been abstracted away or out of reach for the average individual, amateur radio operators have found ways to interact with this technology very closely. Countless miniature amateur/educational satellites are orbiting the earth, carrying out experiments and offering telemetry information to the general public. Amateur radio operators often use special directional antennas and point them to satellites of interest to either communicate with the satellite, or communicate with other radio amateurs using a satellite equipped with a radio repeater.

A problem that arises during this process is that it is not trivial to find the exact location of a satellite at any given moment and point an antenna towards it. We propose a low-cost solution which implements a reasonably accurate rotor that automates this process, as well as a visualization interface that makes it easy to see the global orientation of the antenna and target satellite.

1 Construction

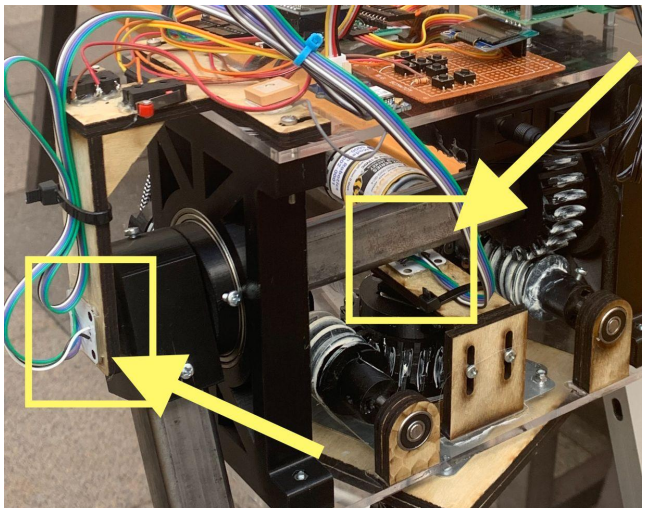
We opted for a custom fabricated rotor assembly that mainly consists of 3D printed and laser-cut parts. These manufacturing methods offer an unparalleled prototype time and cost advantage compared to other traditional methods such as injection molding.



Complete rotor with 2m/70cm Yagi antenna attached.

1.1 Motors, Encoders, and Power

The motors used were the goBilda 5202 planetary gear motors, which offer a 13.7:1 reduction with a no-load speed of 435 RPM at 12V. Rather than rely on the built-in encoders of the motors, we mounted a pair of AS5600 magnetic encoders on each axis, which provided an accurate absolute position reading for the azimuth and elevation axes. The AS5600s were connected over I2C.

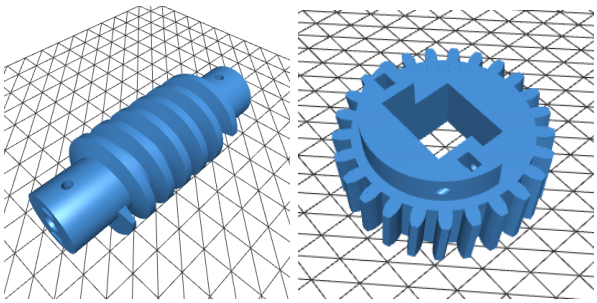


Magnetic encoders (highlighted in yellow box).

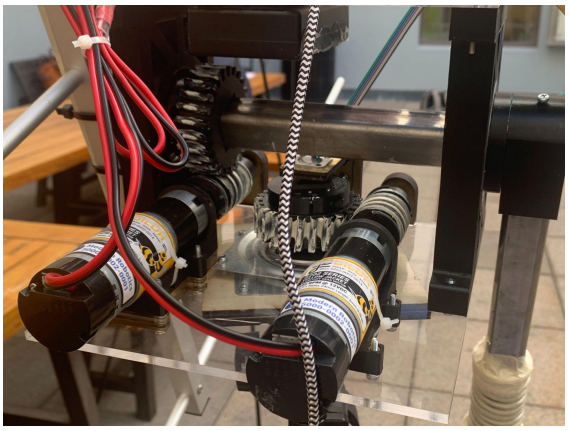
The motors are driven with a pair of DFRobot 15A single DC motor driver boards (one for each motor). Through isolated testing, we determined the typical current draw of our motors to be ~1A each, which is well under the limits of our driver boards. The drivers allow for PWM control of our motors, and we experimentally determined that the elevation axis was able to be rotated without slippage or stalling at a ~30% duty cycle.

1.2 Azimuth/Elevation Axis Assembly

Accounting for some losses, we sized our worm gear ratios assuming a ~400 RPM maximum speed of the motors, which equates to ~1.5 seconds for a full rotation (360 degrees) after a 10:1 worm gear reduction. We decided that this was sufficiently fast while maintaining a high enough torque for our purposes, since a satellite pass would require half a rotation (180 degrees) over the span of ~5 minutes.

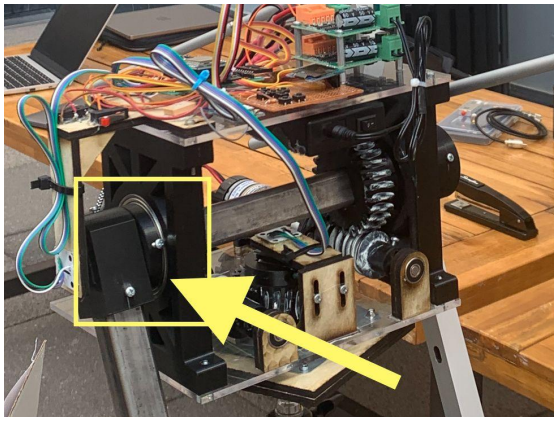


Worm gear / worm wheel design files.



Worm gear / worm wheel attached to motor shaft.

We ensured minimal mechanical slack and durability by utilizing bearings and properly fit inserts for all rotating axes, as can be seen in the picture below.



Elevation axis bearing and insert.

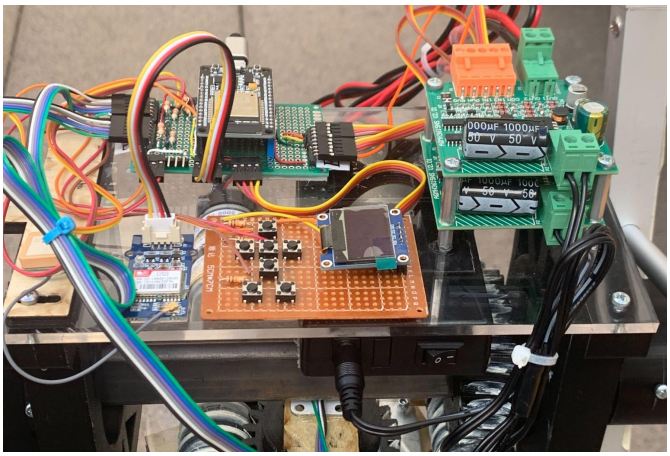
1.3 Antenna & Radio

Considering the fact that most amateur radio satellite contacts occur in the VHF (144-148 MHz) and UHF (420-450 MHz), as well as the relative difficulty of higher GHz range frequency contacts (requiring more specialized equipment), we chose a dual-band (VHF & UHF) Yagi antenna for testing. The antenna features 9.5dB of gain and >12dB of front-to-back ratio, making it highly directional. It features a “gamma” matching mechanism that allows its resonance to be fine-tuned to a desired frequency; we utilized a portable “NanoVNA” vector network analyzer to tune it for the VHF and UHF amateur bands, achieving <1.2 SWR for both bands. We were able to connect any device to the antenna via a coaxial cable—we utilized both an RTL-SDR receiver and a handheld 8W dual-band transceiver for testing.

Due to the significant load that the antenna alone would present to our motor drive mechanism, we mounted a counterweight on the other side of the elevation axis driveshaft to achieve a near-equilibrium state while stationary.

1.4 MCU & Other Electronics

In addition to our motors, encoders, and motor drivers, we also used the ESP32 as our MCU, the Seeed GPS module for GPS (location+time sync data) over serial, and a small OLED display for information such as encoder/target data, time, etc connected over I2C. After testing on a breadboard, the circuit was implemented on a perfboard for increased durability.



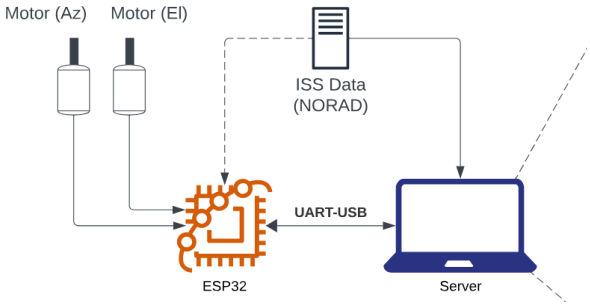
MCU, control board (buttons and OLED screen), motor drivers, GPS, and battery pack mounted on top of the device.

2 System Design

The entire system is designed with the following concepts in mind: state machines, sensors and actuators, and controls. The below diagram shows a rough outline of how our hardware is connected. The ESP32 is responsible for fetching updated TLE[1] data for the ISS and running a prediction algorithm to determine the angles of azimuth and elevation relative to the machine’s current location. It is also responsible for integrating motor controls, GPS data processing, and

[1] See here for more information about the NORAD TLE data format

magnetic encoder readings. The ESP32 connects to a Flask server on a computer via USB, which sends data to the ESP32 and receives the ESP32's calculations to serve to our visualization.



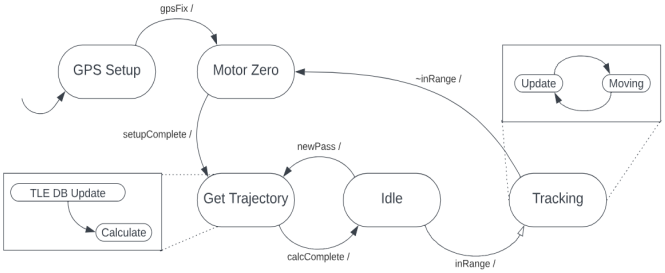
Hardware setup outline

2.1 State Machine Design

Since the ESP32 has to integrate many functions, the best way to represent the different states of processing is a state machine. The state machine has two general phases: setup and processing. The state machine is implemented in `controller.ino`.

2.1.1 Setup. The setup phase is required since the GPS, encoders, and motors need initial calibration and setup before it can function correctly. The GPS has to acquire a fix to a nearby GPS satellite first so the system knows its current location done in `setup_gps()`. Then, we manually move the motors to a "zero" location and find a constant offset bias of the encoders and correctly set the reference point, done in `calibrate_encoders()`. "Zero" for azimuth is set to due North and parallel pointing forward to the ground for elevation. After these steps are done, the machine enters the `PROCESS_TLE` state where it waits for TLE data from the computer to start the first loop of calculations.

2.1.2 Processing. The `PROCESS_TLE` stage involves receiving TLE data for a satellite from the server, predicting the location of the target satellite for the next second in `PREDICT`, and then controlling the motors to point in the desired direction in `TRACK_MOTOR`. After the encoder readings for both azimuth and elevation are within an acceptable range, the server is updated with the latest encoder azimuth and elevation readings in `UPDATE_SERVER` and loops back to `PREDICT` for the next second. The prediction itself is not extremely CPU-intensive, so there is negligible hang time between each second.



State machine for satellite tracking

2.2 Sensors and Actuators

As seen in the pictures above, we used two DC motors to move our antenna in two axes and magnetic encoders to find the absolute position of each motor. We also used a GPS to find the current latitude, longitude and elevation above sea level of the system.

2.2.1 GPS. The GPS is first set up at the beginning on system boot. The GPS has to acquire a fix to a GPS satellite, which will then return valid location data. In the function `setup_gps()`, the GPS data (formatted in NMEA sentences), is continuously read until a valid fix is found. Then, the valid sentence is parsed with a custom written parser (since other packages had trouble parsing our specific Grove Seeed GPS). The longitude, latitude, and altitude are extracted and saved to global variables.

2.2.2 Motors and Encoders. The motors are driven by PWM signals from the ESP32. Controls will be explained in the next section. We used the pair of AS5600 encoders[2] to measure cumulative position. Cumulative position was chosen as it was easier to manage in combination with the negative angles produced from our prediction and to work in a 360 modspace. The angle increases in a counterclockwise direction for the azimuth encoder and clockwise in the elevation encoder. Since the elevation motor's range is restricted by the counterweight and encoder mount, we are limited to a ~270 degree range.

2.3 Controls

First, we attempted to use PID control to control the motors. The code is still there (but commented out). However, because our motors are very powerful and the gears do not have enough reduction, the operable PWM range was between 37 and 40 in a range of [0, 255] before the motor either got stuck or would move too fast for our use case (and constantly overshoot).

[2] We used this library to interact with the encoders

Because of this limitation, we ended up using a naive bang-bang with an acceptable delta of 4 degrees about the desired angle. By setting the PWM to a low value, we can simply rotate the antenna in the closest direction to the desired angle. If the antenna overshoots, it will simply move back in the opposite direction. If it is within the 4 degree range, then the antenna will not move.

This leniency did not affect our overall accuracy, as the bang-bang control combined with a low PWM and precise encoder measurements resulted in a relatively low average deviation from the desired angle (as seen in the Final Results section). Averaging across the range may also have contributed to a relatively close average.

2.4 Prediction

Prediction was done using the SGP4 algorithm[3]. The algorithm takes the current latitude, longitude, altitude, TLE lines and current UNIX time as inputs and outputs the ISS ground latitude, longitude, altitude, azimuth, and angle of elevation.

This prediction is done for every next second, and the rotor moves to the predicted position. On every TLE update from the server, the RTC time syncs with the computer to ensure no drift.

If the satellite is under the horizon (the elevation result is negative), the rotor resumes the starting position and does not track.

2.5 Web Server

The web server has two main functions: call the TLE API[4] and send updates to the ESP32, and receive the ESP32's encoder data to be served to the visualization frontend. This also serves to incorporate networking into the project. This is found in the `server` folder.

2.5.1 Updating the ESP32. The main function is to call the TLE API with any given satellite ID. The API call to `/send_position_to_esp?id=<ID>` will do the following: call the TLE API with the satellite ID (25544 for ISS), format the TLE data and UNIX time in a String, and send it through serial to the ESP32.

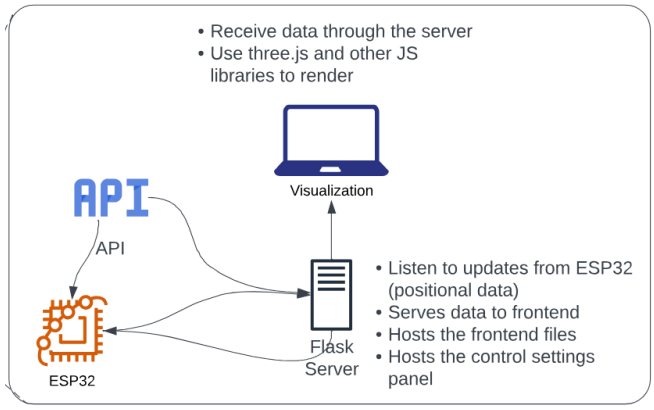
2.5.2 Fetching sensor data. Every time the visualization requests fresh data, it will ask the ESP32 for the current sensor data. The visualization calls `/get_sensor_data` and it will fetch the azimuth, elevation, and longitude and latitude of the ISS (or target satellite), which is then rendered in the visualization. When the ESP32 is not connected, the calculation is performed server-side and sent to the frontend.

[3] See here for the ESP32 implementation of SGP4 we used

[4] This API was used for simplicity

3 Visualization

The visualization allows us to display the movement of the antenna and the ISS in a 3D, interactive environment. The below diagram shows the setup and configuration of the visualization/front-end and server.



Visualization/server stack

3.1 Data Reception

While the flask server handles all of the interaction between the front-end and back-end, we still needed a way to take in and store the data for use on the front-end. The data that the server sends over is a nested dictionary. To make handling that data more efficient and maintain clean code, we created an `ISS` class that has methods to set the azimuth, elevation, ISS latitude and longitude, and antenna latitude and longitude. Our function `getISSData()` fetches the data from the server, creates an instance of the class, calls the appropriate class methods, and returns a list, which allows for easy access, containing the class attributes listed above. The relevant files are in the `web` folder: `visualization.html` and `get_iss.js`.

3.2 Setup & Initialization

We used Cesium to visualize all of the movement. It has a built in earth/globe that users can interact with in a variety of ways, including zooming in/out and rotating. Cesium also makes it easy to upload 3D models. Both our antenna and the ISS models came from NASA. We uploaded the files which returned a unique ID associated with our access token that is used to load the models.

The first step for this process was to call the `getISSData()` function so that we could set up the variables that would later

determine the features of each entity. The ISS contains a position property composed of its latitude, longitude, and altitude. The altitude is hardcoded to 5,000 kilometers, despite its orbit being 400 kilometers. The scale of the ISS is at 50,000. Both of these adjustments were made to enhance visibility and improve the user experience.

We set up the antenna next. The antenna contains a position property and an orientation property. This position property does not contain altitude, as it was not necessary since the antenna is on the ground, and the exclusion of it did not affect the model's accuracy. There is also an orientation property that uses the azimuth and the elevation to point the antenna in the right direction. Importantly, the antenna model that we used had a present orientation, so we had to adjust the azimuth by 2π/3 for correct orientation. The scale of the antenna is 25,000.

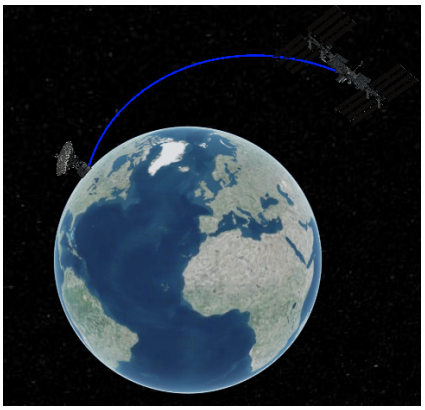
The final part of the setup was adding the line that pointed from the antenna to the ISS. This allowed us to better demonstrate and visualize the antenna's movement and orientation.

3.3 Real-time Rendering

The final part of the visualization was adding in the ability for the position of the ISS, the orientation of the antenna, and the line from the antenna to the ISS.

The line from the antenna to the ISS is updated using Cesium's built in `CallbackProperty()`. This allowed us to set the line's endpoints based on a callback function that took in the latest values of the antenna and the ISS. The `CallbackProperty()` is executed every time the line's position property is accessed, which occurs during the rendering process of each frame. Cesium targets a frame rate of 60 frames per second (FPS).

For the rest of the model, we have an `update()` function. This function fetches the latest data and uses it to update the ISS' position and the antenna's orientation. It utilizes JavaScript's `setInterval()` method, allowing us to call the update function every second.



Visualization of antenna and ISS orientation

4 Final Results

Our rotor was capable of accurately tracking a satellite trajectory; given an up-to-date TLE dataset as well as accurate time synchronization and location data, it was able to calculate the trajectory of the satellite and move when the satellite was passing over its location. A web interface using data from a serial link to the ESP32 was also successfully able to provide a 3D visualization of the antenna and satellite orientation on a model globe.

We validated the accuracy of the prediction and movement by using a consistent test TLE datapoint to first generate a set of predictions with gpredict[5], an open-source satellite tracking program, then using the same TLE data to generate our own predictions. We then cross-checked the azimuth/elevation data points generated by gpredict and the encoder readings taken from our rotor during a test satellite pass. We calculated the average azimuth angle error to be ~0.037 degrees, and the average elevation angle error to be ~0.027 degrees. See Appendix 1 for our test data.

5 Reflection & Possible Improvements

5.1 Mechanical Limitations

We initially determined our worm gear reduction to be correctly calculated, but we found that it was insufficient for our needs. Though the motors were able to rotate the antenna and counterbalance system without problems, it stalled at any PWM duty cycle under 15% and would be too fast with anything over 18%. With a limited operating range, we were not able to use PID control as initially planned and rolled back to a "bang-bang" control scheme. We attribute this to improper gear reduction—either a lower RPM motor or a higher

[5] See more info here

reduction worm gear would have mitigated this issue, but our limited schedule did not allow us to fabricate an improved rotor mechanism.

Due to the mostly plastic construction of the rotor as well as the relatively large size of the antenna being used, there was a lot of play in the movement of our motors. The plastic construction of the gears as well as the lack of a proper mating surface (that would have been improved by introducing a curvature in the worm wheel teeth, increasing contact surface with the worm gear) also contributed to the undesired play. Furthermore, this effect was exacerbated by the relatively choppy motion of the primitive “bang-bang” controller that we were forced to use.

5.2 Limited Testing

Due to the relatively short time we had to test our device in real conditions, we were unable to make a successful contact with a satellite as we planned. When attempting a contact with a satellite, additional considerations such as split operation (different uplink / downlink frequencies) as well as significant doppler shift that exceeds the 2.5KHz bandwidth proved to make operation more difficult than we had anticipated.

5.3 Visualization Design Tweaks

In hindsight, there was no need to create and call class methods for the ISS. Instead, the class constructor could have taken the data in as a parameter and used it to set the class attributes.

Given a larger budget, we could have purchased an antenna model that was more representative of the real antenna we used, which would have been a tad more realistic. The antenna model we have has a base and a dish that both move together. There was no way to only move the dish, which would have been a better option. The only way to do that would have been to recreate the antenna from scratch with the purpose of giving it points that could be animated. However, given a 3D model that more accurately reflected the one we used, this would not have been an issue.

Conclusion

Being able to accurately track not only satellites, but any object in the air for the purposes of establishing a communication link is a useful ability that has many applications. During the development of our rotor, a member of SEB (Space Enterprises at Berkeley)—a student rocketry club here at UC Berkeley—expressed interest in our rotor design; they were working on a similar device for use in tracking their rockets in order to establish a robust telemetry link.

“Working the satellites,” as some radio amateurs call it, also offers the general public the opportunity to explore this technology that has traditionally been closed off to large corporations and government agencies that are able to afford the high cost of operating and utilizing a spacecraft. We believe that the techniques we explore here will with no doubt be useful in a more budget-constrained research / educational / hobbyist setting.

6 Appendix 1

Simulated			Actual (averaged)				
Time	Azimuth (deg)	Elevation (deg)	Time (UNIX)	Azimuth (deg)	Elevation (deg)	AZ error (deg)	EL error (deg)
15:12:01	212.15	0	1702480341	212.17	0	0.02	0
15:12:11	211.76	0.61	1702480351	211.79	0.62	0.03	0.01
15:12:21	211.35	1.25	1702480361	211.37	1.26	0.02	0.01
15:12:31	210.91	1.91	1702480371	210.93	1.92	0.02	0.01
15:12:41	210.44	2.59	1702480381	210.46	2.59	0.02	0
15:12:51	209.93	3.29	1702480391	209.96	3.3	0.03	0.01
15:13:01	209.39	4.02	1702480401	209.42	4.03	0.03	0.01
15:13:11	208.81	4.78	1702480411	208.84	4.78	0.03	0
15:13:21	208.18	5.56	1702480421	208.21	5.57	0.03	0.01
15:13:31	207.51	6.39	1702480431	207.53	6.39	0.02	0
15:13:41	206.77	7.25	1702480441	206.8	7.26	0.03	0.01
15:13:51	205.97	8.15	1702480451	206	8.16	0.03	0.01
15:14:01	205.09	9.1	1702480461	205.13	9.11	0.04	0.01
15:14:11	204.14	10.11	1702480471	204.17	10.11	0.03	0
15:14:21	203.08	11.16	1702480481	203.12	11.17	0.04	0.01
15:14:31	201.92	12.29	1702480491	201.96	12.3	0.04	0.01
15:14:41	200.63	13.48	1702480501	200.67	13.49	0.04	0.01
15:14:51	199.19	14.75	1702480511	199.24	14.76	0.05	0.01
15:15:01	197.59	16.11	1702480521	197.64	16.12	0.05	0.01
15:15:11	195.78	17.56	1702480531	195.83	17.57	0.05	0.01
15:15:21	193.74	19.11	1702480541	193.8	19.12	0.06	0.01
15:15:31	191.41	20.77	1702480551	191.48	20.78	0.07	0.01
15:15:41	188.76	22.53	1702480561	188.83	22.54	0.07	0.01
15:15:51	185.72	24.41	1702480571	185.8	24.42	0.08	0.01
15:16:01	182.22	26.38	1702480581	182.3	26.39	0.08	0.01
15:16:11	178.17	28.42	1702480591	178.26	28.44	0.09	0.02
15:16:21	173.49	30.49	1702480601	173.59	30.51	0.1	0.02
15:16:31	168.1	32.52	1702480611	168.21	32.54	0.11	0.02
15:16:41	161.95	34.4	1702480621	162.06	34.43	0.11	0.03

15:16:51	155.03	36.02	1702480631	155.15	36.06	0.12	0.04
15:17:01	147.43	37.22	1702480641	147.55	37.27	0.12	0.05
15:17:11	139.35	37.89	1702480651	139.47	37.94	0.12	0.05
15:17:21	131.08	37.94	1702480661	131.19	38	0.11	0.06
15:17:31	122.95	37.36	1702480671	123.05	37.43	0.1	0.07
15:17:41	115.28	36.24	1702480681	115.36	36.31	0.08	0.07
15:17:51	108.26	34.68	1702480691	108.33	34.75	0.07	0.07
15:18:01	102.01	32.83	1702480701	102.06	32.91	0.05	0.08
15:18:11	96.52	30.83	1702480711	96.56	30.9	0.04	0.07
15:18:21	91.76	28.77	1702480721	91.79	28.84	0.03	0.07
15:18:31	87.63	26.73	1702480731	87.65	26.79	0.02	0.06
15:18:41	84.06	24.76	1702480741	84.08	24.81	0.02	0.05
15:18:51	80.96	22.87	1702480751	80.98	22.93	0.02	0.06
15:19:01	78.27	21.1	1702480761	78.28	21.15	0.01	0.05
15:19:11	75.91	19.43	1702480771	75.91	19.48	0	0.05
15:19:21	73.84	17.87	1702480781	73.84	17.92	0	0.05
15:19:31	72	16.41	1702480791	72.01	16.45	0.01	0.04
15:19:41	70.38	15.05	1702480801	70.38	15.09	0	0.04
15:19:51	68.92	13.77	1702480811	68.92	13.8	0	0.03
15:20:01	67.62	12.57	1702480821	67.62	12.6	0	0.03
15:20:11	66.45	11.44	1702480831	66.45	11.47	0	0.03
15:20:21	65.39	10.37	1702480841	65.39	10.4	0	0.03
15:20:31	64.43	9.36	1702480851	64.42	9.39	0.01	0.03
15:20:41	63.55	8.41	1702480861	63.54	8.44	0.01	0.03
15:20:51	62.74	7.5	1702480871	62.74	7.53	0	0.03
15:21:01	62.01	6.64	1702480881	62	6.66	0.01	0.02
15:21:11	61.33	5.81	1702480891	61.33	5.83	0	0.02
15:21:21	60.7	5.02	1702480901	60.7	5.04	0	0.02
15:21:31	60.13	4.26	1702480911	60.12	4.28	0.01	0.02
15:21:41	59.59	3.52	1702480921	59.58	3.54	0.01	0.02
15:21:51	59.09	2.82	1702480931	59.09	2.84	0	0.02
15:22:01	58.62	2.14	1702480941	58.62	2.16	0	0.02
15:22:11	58.19	1.48	1702480951	58.19	1.5	0	0.02

15:22:21	57.78	0.84	1702480961	57.78	0.86	0	0.02
15:22:31	57.4	0.22	1702480971	57.4	0.24	0	0.02
						avg az error	avz el error
						0.03734375	0.02703125