UDASH: Server-Controlling Delivery Kart

Louis Cerda, Zahid Safi, Saeed Aljaberi, Jacob Montes, Jonny Palacios-Torres, Zyad Youssef
Computer Engineering, University of Utah, Salt Lake City

***Abstract*—UDash is a food and package delivery service designed for the University of Utah campus. Deliveries are controlled via users connected to the UDash server. The delivery vehicle, Kart, will intermittently receive GPS instructions from our server and navigate toward the requested dispatch area. Upon approximate distance to the pick-up zone, the Kart will notify the server of its arrival via GPS coordinates back to the server.**

I. Introduction

Inspiration for this project comes from a transportation issue on the University of Utah campus. From residence housing to hospital clinics and the engineering and humanities buildings. A total of 203 buildings spanning 1,534 acres of land makes transporting various goods across campus challenging [1]. In addition, many smaller sub-buildings and departments are landlocked, having little access to public roads. Consequently, delivering packages between facilities requires delivering on foot or driving vehicles slowly to avoid pedestrians. Both solutions are time-consuming and require manually transporting the packages between buildings.

UDash is an online delivery service where customers can request and deliver various goods across campus through an online website. The delivery Kart will dispatch deliveries based on popular routes between the different zones, as shown in Fig. 2.

As shown in Fig. 1, a typical delivery and demonstration will involve a user-controlled route between the Merrill Engineering Building (MEB) and Warnock Engineering Building (WEB). Each building's entrance will serve as the designated zones for dispatch. Assuming that the Kart starts at MEB, a patron will request delivery to WEB through our website. The UDash Website will then communicate with the server, which is responsible for handling the orders and contacting the Kart. The Kart begins by receiving the first set of GPS coordinates that serve as directions for the vehicle.

During the delivery, the Kart will frequently ping its current location back to the server, and the server will validate if the Kart is operating correctly. The delivery is complete when the Kart arrives at a zone by an approximate distance. The user will see it's final location through GPS coordinates on the server.

The Kart is no bigger than 22 inches wide and 21 inches tall operated via micro-processor. Customers will recognize the vehicle by its distinct U shape that is synonymous of our university's logo. Packages will be seated in the center of the U and will controlled remotely by a human operator.

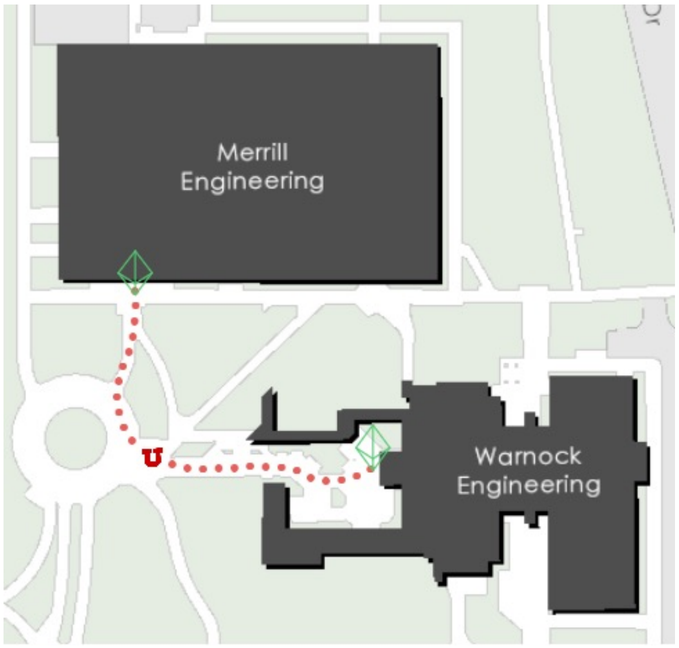

Fig. 1. The proposed working route. Delivery will be made between Merrill Engineering to Warnock Engineering.

II. Background

The approach to UDash is separated into three components, hardware, software, and firmware. Hardware will consist of the RC Kart and sensors. The software comprises the server and website responsible for controlling the Kart manually. Lastly, firmware involves the GPS hardware and establishing a connection between the Kart and the server.

A. Hacking the RC Kart

The hardware team uses Arrma Senton 4x4 V3 Mega 550 as UDash's Kart platform (Fig. 3). Arrma's Karts are durable, allow for chassis adjustability and replacement parts, and support a broader wheelbase to support the U frame, shown in Fig. 4. It has a top speed of 35 mph and an estimated 10 lbs carrying capacity [2]. The carrying capacity is the project constraint as the extra weight of electronics and the frame itself and package will factor into the total weight of the Kart. The different speed provides additional torque needed to carry the weight Kart across campus but not exceed dangerous rates. The Kart is controlled through a radio transmitter and receiver

B. Software

A server and user interface (UI) is used to help communicate with the Kart and different users for our project. A web

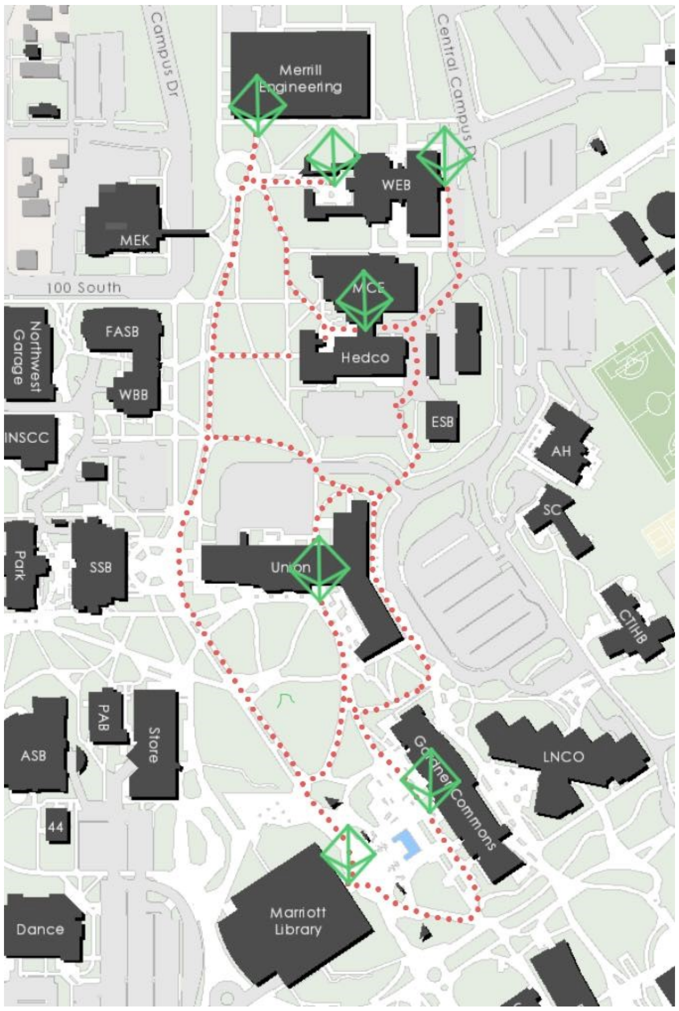

Fig. 2. Map indicates how a sophisticated delivery system could be implemented on campus. The red dotted lines indicate possible routes taken by the Kart, the green polygons indicate delivery zones. Original image taken from the University of Utah campus and was later annotated [1].

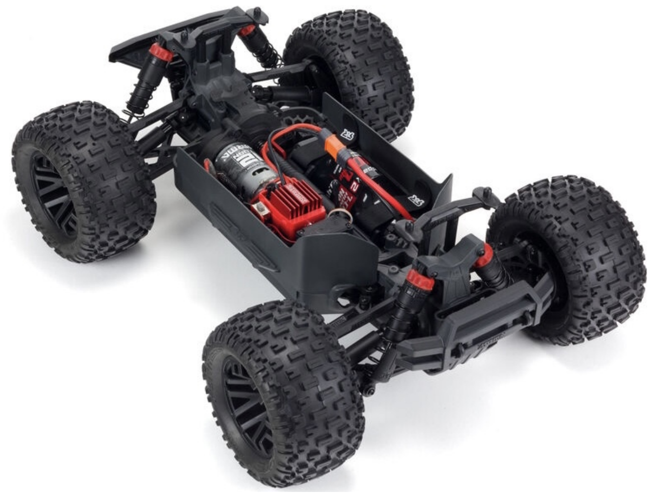

Fig. 3. Arrma Senton 4x4 V3 Mega 550 kit with the plastic top body removed used for the Kart [2].

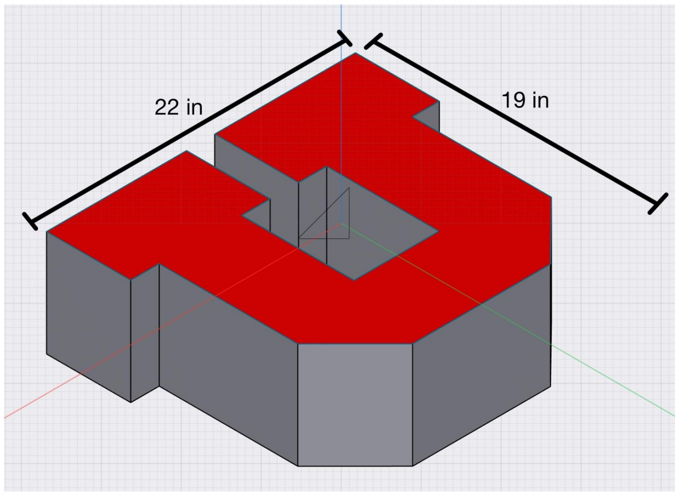

Fig. 4. 3D printing model of the Block U designed using the iOS Shapr3D application.

server with application programming interface calls (APIs) is commonly seen in similar web projects. Websites like Google and Netflix utilize API calls to interact with their website based on their application's needs. Google, for example, uses API calls that connect users to google maps and various services tied to their google account. A similar design approach was used for our website, where different API calls can track the Kart's usage and location based on API calls made to the website.

The UDash project takes inspiration from DoorDash's delivery service model. The UDash website allows the user to control the Kart, get a live video feed, and see the GPS location of the Kart. The UDash website will employ distinct API calls to forward movement requests to the Kart, while the Kart itself will utilize similar API calls to return GPS data. Not to mention that the video stream will be hosted on the Kart itself, allowing the operator to view what the Kart sees.

C. Firmware

To navigate between different drop off-points on campus, a consistent protocol was constructed for transmitting and receiving the current location of the Kart. To select the best Global Positioning System (GPS) device for the project, some background in GPS systems was conducted to minimize potential risks in the engineering process. Using a GPS, the positioning of an object can be calculated using *trilateration*, a mathematical approach to finding an object using two-dimensional or three-dimensional coordinates after connecting to a minimum of 3 satellites [3]. To begin sending GPS coordinates to the server backend, the GPS has to have a set of reliable features that will benefit the kart as follows:

- Fast Initialization: The Kart will be highly dependent on knowing its location on startup to know where to go.
- Accuracy: All GPS modules are not 100% exact, but

having an overall accurate GPS will help pinpoint any round-off error.
- Reliable Connection: Minimizing the loss of GPS satellite connections is crucial to ensure position updates occur at their required times.
- Ease of Use: A GPS with dedicated server libraries to build off from and easy hardware connections.

After considering all the design factors, the GlobalSat BU-353-S4 USB GPS Receiver was chosen for its reliability for current and previous users, a hot-start time of one second (35 for a cold-start), a 10m 2D Root Mean Square (RMS) for accuracy, a WiFi to USB connection with a dedicated GPS library called *GPSd* that can parse data in JSON.

Using the Raspberry Pi 4, the GPSd library will read latitude and longitude coordinates, which will then be stored as the Kart's central transportation controller. By reading the coordinates, the Raspberry Pi can calculate the current position of the Kart and compare the current location with the database. The database tells the Kart where to go after the data is sent to the server.

III. Work Implemented

The team proposes to deliver packages between MEB and WEB. A data set of GPS coordinate points will mark the proposed route. In addition, UDash will have two pick-up zones at the entrance of each building. Customers will then be able to request pick or delivery from the two locations from the UDash Website.

The team has been able to create a working kart in which we can communicate with using a website through an API back end to send movement commands, GPS coordinates and a video stream. The Pi, which is the central component of the kart uses our phone hotspot in order to send the video stream and communicate with the API. Since the usage of API for controlling the kart and a working video stream, we don't need to be near the Kart and can control it remotely.

A. Hardware

The Arrma RC Kart operates by receiving radio signals emitted by the remote controller to the SLR 300 Antenna Receiver. The signal is then sent through the receiver plug, which is connected to the Electronic Speed Controller (ESC). The ESC is an electrical circuit responsible for controlling and regulating the DC motor's speed through two pulse-wide modulations (PWM) wires. The first PWM wire is responsible for managing the speed of the motor, while the second PWM wire acts as a traditional servo motor signal that controls the Kart's steering. The Hardware team used this knowledge to their advantage by intercepting the Kart's communication with the motor using the PWM signals from the onboard Raspberry Pi 4 rather than the Kart's internal radio receiver.

To fully use the Raspberry Pi's PWM signals, the hardware team had to first measure the incoming PWM signals from the receiver using an oscilloscope. The PWM signals were recorded for both steering and throttling, as shown in Fig. 5. Different PWM signals were recorded for the PWM signals so the Kart could change speeds and move in reverse. The main issue with this task was that the Kart had its communication protocol with the remote control, and there needed to be publicly available documentation for the operating frequencies of the PWM. In addition, testing was conducted to fine-tune the signals from the Raspberry Pi to work correctly with the Kart.

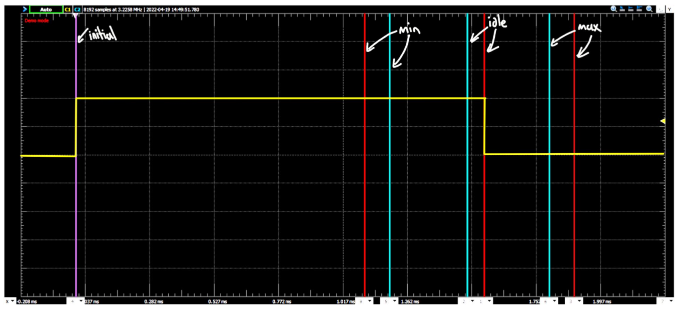

Fig. 5. PWM signals of the Kart for server control.

Once the correct PWM signals were determined, The Raspberry Pi 4 is then directly connected to the ESC for PWM emulation. Python libraries were created to control the speed and operation of the RC Kart. Various high-speed libraries and commands were needed to emulate the Kart's movements (left, right, accelerate, brake/reverse). The library was essential for the project because it prevented the kart from making adjustments to the PWM signals in the same part of the code where GPS and server responses were occurring.

The python libraries added a layer of abstraction that allows other portions of the code easier access to the PWM signals. For example, this allowed the firmware to be in charge of the GPS and the client to call functions such as right (45 degrees), left (45 degrees), straight (50% power), etc., without knowing the exact duty cycle for each PWM signal.

Since many of the events coincide, which can cause several issues such as input lag, unwanted delays, and connectivity problems. For these reasons a Raspberry Pi 4 for its microprocessor is used because it allowed for faster calculations and mitigation of the issues mentioned before. In addition, the Raspberry Pi 4 lets the Kart use an ultrasonic sensor for proximity detection so that the Kart can stop within milliseconds in an emergency.

Another issue is connectivity and ensuring the Kart stays connected to the server. To address this issue, the Kart uses the mobile WiFi hotspot functionality from an iPhone 13 Max Pro to continuously connect the Raspberry Pi to the Internet. This circumvents any issues the Kart would have had from being out of range of the school's WiFi router, such as disconnecting mid-delivery. It also allowed for the Raspberry Pi to consistently have a strong signal close by as they are known to latch onto WiFi signals for as long as possible despite the strength of the WiFi signal. Weak signals would have caused issues with the Karts transmissions to the server, this mobile hotspot solution mitigates any issues that could have occurred between Kart and the server.

B. Firmware

To simultaneously switch from pinging the server the Kart's current location to receiving commands from the user. The Kart had to be implemented to run asynchronously via (python's asyncio libraries). The async functionality allows the Kart to be in a non-blocking fashion and execute commands after a specific timer has been reached. Shown in Fig. 6 shows the overall design of all communication components for the UDash Kart.

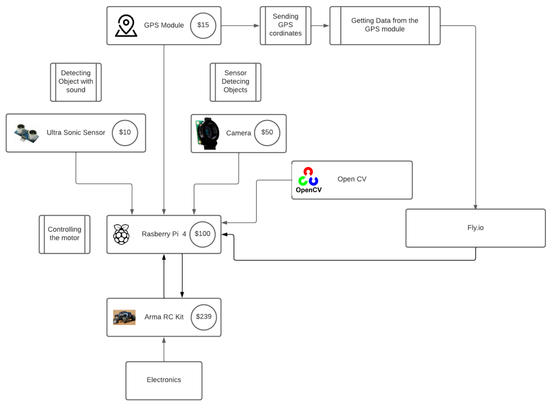

Fig. 6. The overview of the required source connections to have a fully functional server-controlled Kart.

1) Kart Client: The added flow allows the Kart to send GPS coordinates every 4 seconds. Then, it processes the driving commands sent to the C GPIO libraries that communicate with the Raspberry Pi 4's GPIO pins to signal what the Kart should be doing (driving, resetting, reversing, etc.). Shown in Fig. 7 is the flowchart of what the Kart determines to do next after specific actions.

2) Ultrasonic: The ultrasonic sensor is used to help the Kart avoid obstacles. It's a device that measures the distance between the sensor and an object in its path. The sensor has two components: transmitter and receiver powered using a voltage divider of 3V (in reference to the Pi's 5V output). This device emits a sound wave and waits for it to be reflected in the receiver.

The sensor calculates the distance of an object and notifies the Kart if there's an object blocking its path. If the Kart finds an object less than 10μs away, the Kart will no longer progress forward if the user tries to accelerate forward. The object needs to leave the surrounding detection area to drive forward again. The distance of an object from the Kart is calculated using the following equation,

$$D = \frac{1}{2} * T * S \tag{1}$$

where D is the distance, T is time, and S is the speed of sound.

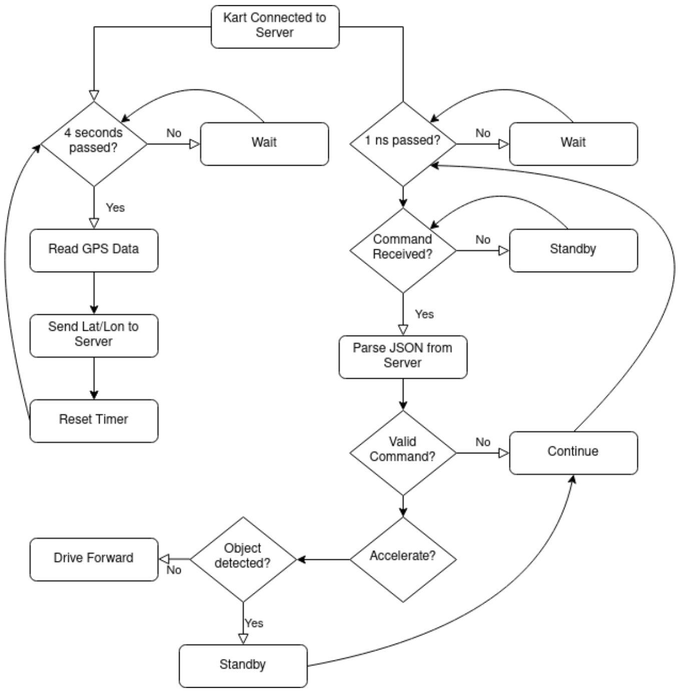

Fig. 7. Kart client - A high-level overview of the logic to run the Kart.

3) Live Stream: The Kart is being controlled remotely by keyboard inputs sent from our website, as well as Arducam (live stream to monitor) to move the Kart to the delivery location remotely. Arducam provides methods to the stream such as, return the frame, and stopping the stream entirely.

The website uses flask API to live stream the video. With flask API, the website can stream the video on a local host with no latency and reasonable resolution, which is relayed to the website for viewing.

C. Software

As a delivery service, our primary concern was how well the Kart takes in orders and notifies them in which direction it should go. So, to make things simple, the software team created a user-friendly website using RESTful (Representational State Transfer) API calls that aids in the communication process between the Kart and the server, seen in Fig. 8.

The website acts as a front-end graphical user interface, allowing the user to control the Kart through W-A-S-D keyboard keys, while also supplying the user with a live video feed streamed straight from the Kart, enabling us to see what the Kart sees. In addition, the website can communicate the Kart's location prompting the users with information about the Kart's last site.

To create a responsive server, the server uses FastAPI, which allows efficient development of web server and reduced the number of files and configuration settings that generally come with web frameworks like Django. First, to run the back-end, FastAPI and Unicorn were used for the server. Next, API endpoints were created that enable the Kart to communicate with the website and vice versa.

After looking into multiple online documentations, the software team had decided to utilize FastAPI for the back-end while utilizing the React.js framework, and Javascript for the front-end [4] React.js simplified the API calls from the back-end, making returning the data much more accessible, especially with the website's user-friendly interface. Finally, everything was tested locally to ensure a proper connection was established. The server utilizes Fly.io for continuous operation, making it easier for both the website and Kart to communicate. The following diagram represents the overall software architecture:

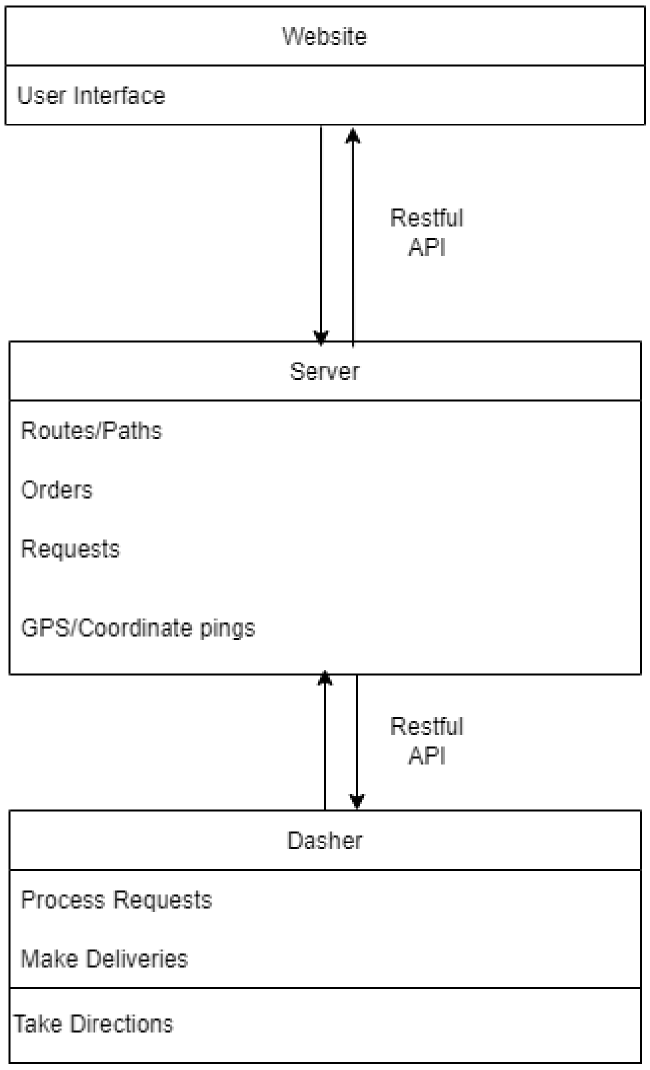

Fig. 8. The UDash server architecture to send commands to the Kart client.

IV. Server Frameworks and Libraries

- **ReactJS** – Used to design and implement the look of the website and also to make it user friendly
- **Fly.io** – Used to deploy our Back-End server so that it is accessible by the website and the Kart
- **FastAPI** – Used to implement the Back-End API's that are used by the Kart and website
- **Docker** – Utilized for software scalability and deployment
- **Google Maps API** – Used within the website to display the Karts location
- **ngrok** – Utilized for exposing and hosting the Kart's video live stream

References

[1] "Campus map - the university of utah," 03 2022. [Online]. Available: https://map.utah.edu/

[2] "Arrma senton mega radio controlled car - designed fast, designed tough," www.arrma-rc.com. [Online]. Available: https://www.arrma-rc.com/rc-cars/latest/senton/mega/4x4

[3] P. Sirish Kumar and V. Srilatha Indira Dutt, "The global positioning system: Popular accuracy measures," *Materials Today: Proceedings*, vol. 33, pp. 4797–4801, 2020.

[4] React, "Getting started – react," Reactjs.org, 2019. [Online]. Available: https://reactjs.org/docs/getting-started.html

[5] "Sql (relational) databases - fastapi," fastapi.tiangolo.com. [Online]. Available: https://fastapi.tiangolo.com/tutorial/sql-databases/

[6] Z. Kang, B. Tapley, J. Ries, S. Bettadpur, and P. Nagel, "Impact of gps satellite antenna offsets on gps-based precise orbit determination," *Advances in Space Research*, vol. 39, pp. 1524–1530, 01 2007.