



**Politecnico
di Torino**

Politecnico di Torino

Facolta di Ingegneria

FLIGHT CONTROLLER

Master degree in Electrical Engineering

Author: Farah Goachem

Professor Claudio Passerone
in collaboration with Motion Engineering

March 29, 2025

Contents

1	Dedication	1
2	Introduction	2
2.1	Aim	2
2.2	What is Kite Energy?	2
2.3	General description	5
2.4	Functioning	6
2.5	'Eight' mode Flight	10
2.6	Next steps	13
3	Mathematical model	14
3.1	Context	14
3.2	System architecture	16
3.3	Reference frame	18
3.4	Flight Controller components	18
3.5	Reference frame	19
3.6	Conversion	19
3.6.1	Kite position in the wind window	21
3.6.2	Rotation matrix	21
3.6.3	Position of the kite in global reference system	22
3.6.4	Multiplication	23
3.6.5	Cart2Pol	23
3.6.6	Derivative	23
3.6.7	Rotation matrix	24
3.6.8	Gamma calculation	24

3.7	Supervisor	26
3.7.1	Flight phase detection	26
3.7.2	Sign computation	27
3.8	Trajectory Control	27
3.8.1	Gamma controller	27
3.8.2	Theta Controller	30
3.9	Throttle Control	32
4	Graphics	33
4.1	Cubical trajectory	33
4.2	Graphics server/client implementation	35
4.3	Trajectory implementation with spherical trajectories	37
4.4	Simulation trajectory visualization	38
4.5	Camera	41
5	Hardware	45
5.1	PC WINDOWS general purpose	45
5.2	IPC Siemens	47
5.3	HARDWARE and Interconnections	50
5.3.1	DLL	51
5.3.2	Graphics	54
6	Simulation and future steps	55
6.1	Simulation	55
6.2	Future steps	58
6.3	Collaborations and Acknowledgments	58
7	Appendix	59
7.1	ENU reference frame	59
7.2	UDP and TCP	60
7.3	DLL	64
7.4	PID	65
7.5	Simotion	66
7.6	PROFINET	68
7.7	Socket Programming	70

8 References**73**

CHAPTER 1

Dedication

This thesis is for all the amazing people who have supported me along the way. To my family, for always being there with love and encouragement, and to my friends, whose faith in me has never wavered and has kept me going. A big thank you to my colleagues for all the collaboration and motivation. And finally, to everyone who's played a part, big or small, in the journey of knowledge and growth—this work is for all of you.

CHAPTER 2

Introduction

2.1 Aim

The aim of this project is the development of software for an industrial PC, based on Simulink/Matlab simulations, designed to create an automatic flight controller for high-altitude wind turbines. The wind generator utilizes a large sail ("Kite"), tethered by two ropes that drive two motor generators in a ground station, the energy produced by the system will be captured and stored in batteries for future use.

Currently, the system is manually controlled via a joystick. The goal is to automate the entire system, transitioning from manual control to fully autonomous operation.

The software will interface with various sensors on board the Kite, which are used to measure acceleration, altitude, force, and cable length. These sensors communicate wirelessly with the ground station using appropriate transmission protocols.

2.2 What is Kite Energy?

Kite energy exploits wind energy through kites. Unlike traditional wind turbines, which have rotating blades mounted on towers, kite energy is based on large kites that fly at high altitudes, where winds are stronger and more consistent. These kites are attached to special cables that, by moving in the wind, drive a generator and produce electrical energy.

The main advantages of this technology include the use of winds at higher altitudes, less visual impact, and a lower land footprint.

Kite energy is particularly well-suited for regions with strong, stable winds at high altitudes. These areas allow kites to capture winds that are more powerful and consistent than those found at the Earth's surface, where traditional wind turbines might not operate as efficiently. For example, coastal

areas often experience constant and strong winds, making them ideal for kite energy. In these regions, kites could fly tens of meters above the sea, where the wind is less disturbed by terrestrial obstacles like hills or buildings. Similarly, desert or plain areas, which are often free from trees and other hindrances, also provide stable wind conditions, perfect for kite energy systems. High mountains or hills are another ideal setting, as wind speeds tend to increase with altitude, allowing kites to harness energy from winds at great heights.

Kite energy also proves advantageous in areas with limited space or difficult access. Traditional wind turbines require tall towers and substantial infrastructure, which can be costly to build and difficult to install in remote or hard-to-reach locations. Kite energy, however, requires fewer large ground structures and can be more easily deployed. This makes it particularly useful in areas such as small islands or islets, where building traditional wind turbines is impractical. Similarly, for mountainous or otherwise inaccessible regions, where transporting and assembling traditional turbines is expensive, kite systems could offer a more feasible alternative.

In urban or industrial environments, where space is limited and traditional wind turbines may not be viable, kite energy can offer a solution. Despite the density of buildings, kites can be mounted on existing structures, such as platforms, rooftops, or even on industrial sites, without occupying additional land. For instance, parking lots or industrial sites could host kites flying above buildings or structures, collecting energy without interfering with ground activities. This discreet method of energy generation could prove valuable in areas where minimizing the visual impact and optimizing space are key concerns.

One of the key benefits of kite energy is its efficiency in harnessing high-altitude winds. While winds at ground level can be turbulent and inconsistent, the winds at higher altitudes, where the kites fly, tend to be more constant and stronger. This allows kites to generate more energy compared to traditional wind turbines, which may not perform as effectively in less stable wind conditions near the ground.

Another significant advantage of kite energy is its low visual impact. Traditional wind turbines, with their tall towers and rotating blades, often create a noticeable visual presence over large areas, which can be a concern in many landscapes. Kites, on the other hand, are far more discreet since they fly at high altitudes and don't require large, imposing structures. This makes them a promising option in locations where preserving the natural aesthetic is important.

Kite energy also offers low infrastructure costs. Unlike traditional wind turbines that need tall towers and complex infrastructure, kite systems primarily require the kite itself and a cable system. This dramatically reduces initial installation costs. Additionally, since kites occupy less land space, they don't require vast open areas like wind turbines do, making them more suitable for locations with

limited space.

Lastly, kite systems are more flexible and adaptable than traditional wind turbines. They can be easily adjusted to work in a variety of environments—from the open sea to remote, hard-to-reach areas. Kites can also be deployed on a smaller scale, such as in community-based energy systems or to power smaller structures, offering versatility in how and where they are used.

Despite its promising potential, kite energy remains an emerging technology. As a relatively young field, many of the practical solutions are still in development and undergoing testing. The systems currently face several challenges, such as ensuring the durability of the kites, managing the cables effectively, and addressing safety concerns. Furthermore, long-term reliability is another key factor that needs to be fully addressed before kite energy can be deployed on a larger scale.

Another limitation is vulnerability to weather conditions. While kites are designed to operate in strong winds, extreme weather events such as storms can cause significant damage to the systems. Adverse weather could not only impair the functionality of the kites but also temporarily reduce energy production, making it less reliable during severe conditions.

The safety concerns associated with kite systems are another drawback. Since the kites are tethered to long, high-tension cables, managing these cables can pose risks. Issues such as cable wear, tension buildup, or potential failures in the control systems could create hazards, both for the equipment and the surrounding area.

Finally, while kite energy is efficient in high winds, it still remains an intermittent energy source. The production of energy depends on the wind conditions, and factors like wind speed and direction can cause fluctuations in energy generation. This unpredictability can limit the stability and consistency of the power produced, which makes kite energy less reliable compared to more stable energy sources. So Kite energy offers a promising solution in specific contexts where other forms of renewable energy might face challenges. It is especially useful in areas where winds are strong and stable at high altitudes, such as coastal regions or mountainous terrains, where traditional wind turbines might not be as effective. Additionally, kite energy can be deployed in remote or hard-to-reach locations where the installation of large wind turbines would be difficult or too costly.

Another advantage of kite energy is its suitability for areas with limited land space, such as urban environments, industrial sites, or small islands. In these settings, where land is at a premium, kite systems can provide a lightweight, non-invasive energy solution without requiring vast open areas. Furthermore, for regions where visual impact is a concern, kite energy provides a more discrete option. The kites operate at high altitudes, so they do not have the same visual footprint as traditional wind turbines, which can be imposing in certain landscapes.

In conclusion, while kite energy is a technology with great potential, it is still in the developmental stages. It could be a game-changer in areas where traditional wind turbines aren't effective or feasible. However, it's important to consider its current limitations, including the technical challenges it faces, and the need for further innovation to ensure that kite energy can deliver reliable and consistent power.

2.3 General description

In this section, the overall structure of the system will be detailed. As mentioned in the previous chapter, the system utilizes airfoils, or power kites, to capture wind energy at altitudes ranging from 250 to 1000 meters above ground level. The energy harnessed is then stored in electrical accumulators. The system is managed and controlled by a machine called the KE60, which was designed and developed by Kytenergy Corporation (for more information, it's possible to visit their website at <https://kitenrg.com/>). A visual representation of the KE60 is shown in Figure 2.1, and its components will be further explained in this section.

The airfoil is connected to the actuation unit located on the ground via two composite fiber cables, wound on two spools which are in turn mechanically connected to two motor-generators. A ground control unit automatically guides the airfoil by acting differentially on the two cables.

The machine generates electricity by converting the pulling forces on the cables into electrical power. The operating cycle is composed of two phases: a traction phase and a recovery phase. During the first phase the airfoil is guided along trajectories that cross the wind, generating large forces that unroll the guide cables, causing the spools to rotate. In this phase the motor-generators produce electric current. In the second phase the cables are rewound on the spools spending a fraction of the energy stored in the traction phase.

The machine consists of:

- a chassis with wheels, equipped for towing and equipped with a braking system, lighting system and extendable stabilisers, on which all the other components are fixed,
- a pair of spools on which the composite fiber cables are wound,
- a pair of motor-generators connected to the spools, which rotate the latter in the recovery phase and which generate electrical energy in the traction phase,
- a pair of motors that translate the spools along linear guides to facilitate the orderly winding and unwinding of the cables,
- a cable exit system, composed of a series of pulleys and ending with a rotating and oscillating arm which allows the correct direction of the cables and the dampening of force peaks,

- a series of electrical cabinets that house the power electronics and control electronics of the system,
- a cabinet that houses the battery pack (batteries + capacitors) in which the energy produced is accumulated.

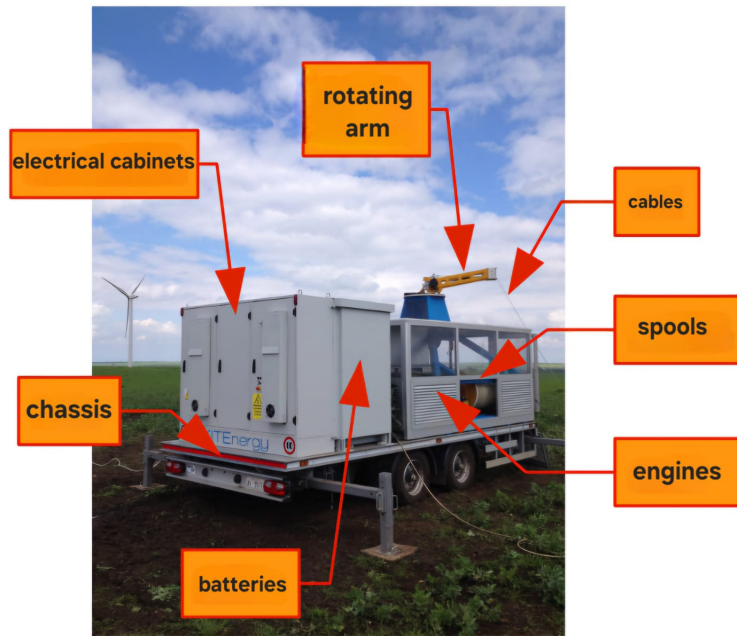


Figure 2.1: KE60 Image

The accumulators are located inside the compartment situated between the electrical cabinets and the cabin. These consist of 33 lead-acid batteries, each with a voltage of 12V and a capacity of 75Ah, arranged in series to achieve a total voltage of approximately 400V. These batteries play a crucial role in ensuring the system's efficiency by storing the energy generated by the kite, which can then be used for continuous operation or to power various components of the system when needed.

2.4 Functioning

The KE60 is equipped with an operator cabin that features an interface for interacting with the system. This interface includes controls for operating the kite and provides real-time machine status information through an intuitive Human-Machine Interface (HMI) displayed on a monitor.

When the kite is in flight, all machine operating data is continuously monitored, displayed on the

user interface, and, if logging functionality is enabled, recorded in the log file for further analysis. These data are primarily used for more in-depth analysis. Based on the collected information, a mathematical model will be developed to simulate the system's behavior, optimizing performance and predicting potential future scenarios(see chapter "Next Steps"). This model will assist in improving flight management and identifying the most influential variables in the kite's operation.

In the current system, flight control is primarily managed through the joystick lever, which adjusts the differential between the two cables for turning, and the throttle, which regulates the winding and unwinding of the cables on the spools(a picture of the lever and throttle is in figure 2.2). By manipulating the joystick lever, the user alters the differential between the cables, affecting the airfoil's turning behavior. When the lever is in the neutral (center) position, the differential is zero, and the airfoil maintains its current orientation.

Pushing the lever to the left creates a differential that causes the airfoil to turn left, while moving it to the right induces a differential that makes the kite turn right. The amount of differential is directly proportional to the movement of the joystick lever.

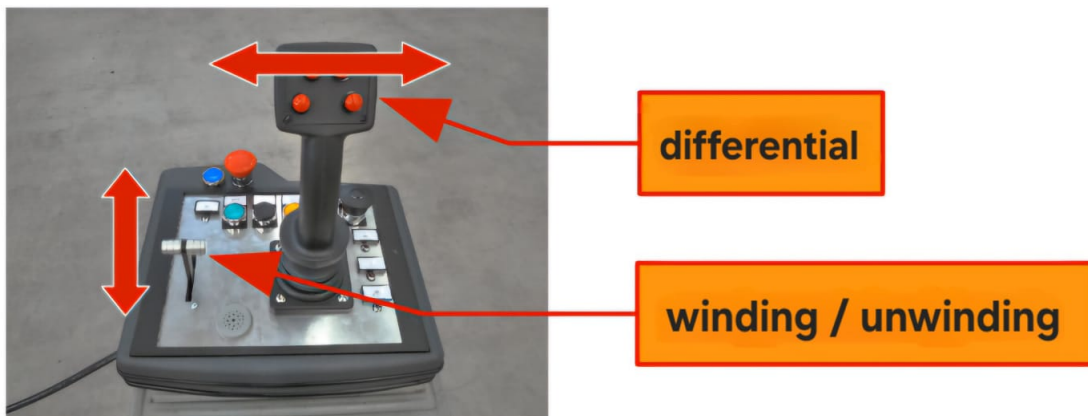


Figure 2.2: Actual joystick used to drive the kite

The differential value corresponding to the end positions of the lever can be set via the following input field on the HMI:

“Max differential” - defines the maximum value of the differential in meters, corresponding to the end position of the joystick.

The position of the throttle directly determines the rotation speed of the spools. Moving the handle

forward from its central position rotates the spools in the direction of cable winding, while moving it backward rotates the spools in the opposite direction. The rotation speed is proportional to the amount of throttle movement.

The speed values corresponding to the throttle end positions can be set via the following input fields on the HMI :

“Max positive speed” - it defines the maximum cable deployment speed in m/s, corresponding to the end-stop position of the throttle lever forward, meanwhile **“Max negative speed”** - defines the maximum cable winding speed in m/s, corresponding to the end-stop position of the throttle lever backward.

The terms positive and negative are used to distinguish the two opposite directions of motion:

a positive speed corresponds to cable deployment (unwinding), as the cables extend outward,

a negative speed corresponds to cable winding (rewinding), as the cables are pulled back onto the spools.

This convention helps clearly define the system’s behavior and allows for proper control of the spool rotation direction.

The fields that are used to define and configure the command parameters are clearly illustrated in Figures 2.3 and 2.4, providing a comprehensive overview of the system’s control settings.

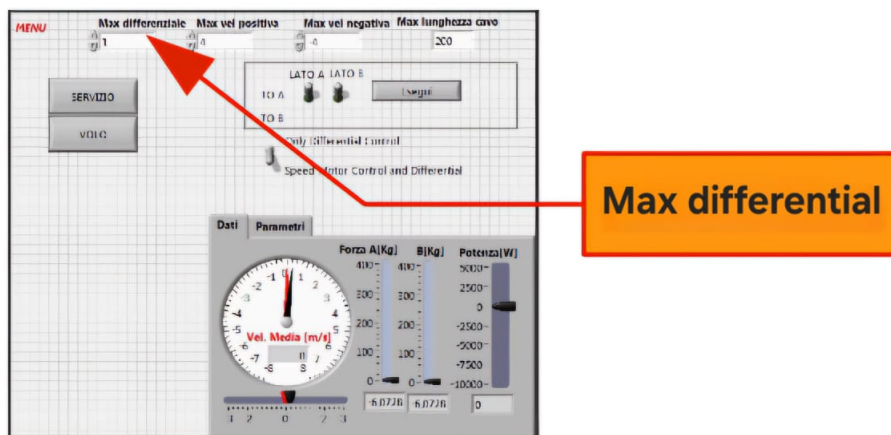


Figure 2.3: Max differential

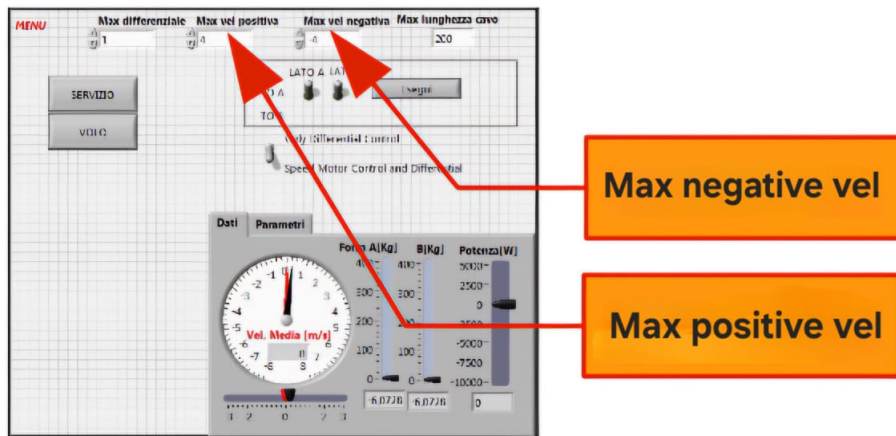


Figure 2.4: Max velocity

Moving the throttle lever forward from its central position, the applied torque is in the cable deployment direction; moving it backward, the applied torque is in the cable winding direction. Note that the direction of the applied torque does not directly determine the rotation direction of the spools, as this also depends on the force exerted on the cables by the wing profile. The torque applied to the spools is proportional to the amount of throttle lever displacement.

The torque values corresponding to the throttle end positions can be set via the following input fields on the HMI:

“Max positive speed” - defines the maximum torque value in the direction of unwinding in Nm, corresponding to the forward throttle end position meanwhile **“Max negative speed”** - defines the maximum torque value in the winding direction in Nm, corresponding to the end position of the reverse throttle.

Furthermore the system includes a dedicated button that enables users to save logs of various data. This functionality ensures that recorded information can be securely stored and accessed later for in-depth analysis and troubleshooting (see Figure 2.5)

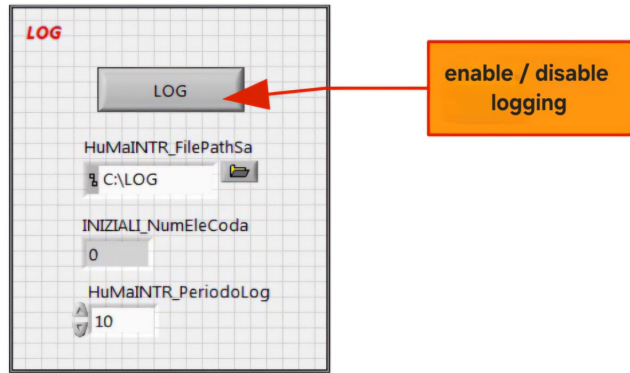


Figure 2.5: Logging button switch

2.5 'Eight' mode Flight

The main objective of this experiment is the generation of electricity by exploiting wind energy.

The critical phase of the experiment is the reentry phase: that is, when the stored energy is consumed. It is therefore necessary to devise a mathematical model that minimizes energy consumption while maximizing the energy created by the Kite.

This model was formalized by Professor Fagiano (Polytechnic University of Milan), who demonstrated that by making the kite move in the shape of a 'eight', the consumption is minimal. The 'eight' mode is a flight trajectory in the shape of the number eight, used in Airborne Wind Energy (AWE) systems to efficiently extract energy from the wind. The distinctive feature of this mode is that the aircraft—typically a kite—operates in a cycle that includes both ascending and descending phases. During the descent, the aircraft can generate significant traction without the need for additional energy input, thanks to the combination of lift and drag forces. In this trajectory, the aircraft performs two opposite semicircles that form the figure-eight pattern. The descending phase is particularly important because it allows the system to recover energy, creating a cycle that maximizes efficiency.

Phases of the "Eight" Flight

Initial Ascent (first ascending semicircle):

In the first part of the trajectory, the aircraft rises upwards, harnessing the lift generated by the wind. In this phase, the aircraft is typically oriented upwards, usually with an angle of attack between 30° and 45° relative to the wind direction. The upward motion requires energy, which can come from

the reel-out speed (for ground-based systems) or from an onboard engine. However, this energy is recovered in the subsequent phases of the flight.

Peak and Transition to Descent (top of the path):

Once the aircraft reaches the highest point of the trajectory (the "peak"), it begins the transition to the descent phase. During this phase, the flight speed may decrease, but the aircraft remains capable of generating traction due to the wind energy. The point at which the aircraft changes direction towards the downward trajectory is crucial: the angle of attack and control over aerodynamic forces determine how efficiently the transition occurs.

Descent (second descending semicircle):

In the descending phase, the aircraft follows an inclined path downwards, utilizing drag to increase the traction. In this phase, the drag force is maximized, and since the aircraft is moving downward, it generates significant energy recovery. The downward motion takes advantage of gravity and aerodynamic forces to produce power without external energy input. This is one of the key advantages of the "eight" mode: the aircraft does not need energy to maintain flight during the descent and can instead produce energy efficiently.

Recovery Phase and Cycle Repetition:

After completing the descent phase, the aircraft returns to the starting point of the cycle (i.e., the initial ascent point). Here, the next iteration of the figure-eight path begins, continuing the cycle without the need for external energy input.

Aerodynamic Forces in the "Eight" Flight

The flight in the "eight-down" mode relies on the interaction between several aerodynamic forces to optimize energy production:

Lift:

During the ascent, the aircraft generates lift to rise. Lift is a force perpendicular to the relative wind and depends on the shape of the wing, the wind speed, and the angle of attack. Lift is essential for keeping the aircraft in the air and for allowing it to ascend at the beginning of the cycle.

Drag:

During the descent, drag becomes crucial for extracting energy from the system. Drag is the resistance of the air to the aircraft's motion and increases with speed. In the descending phase, drag becomes a useful force, as it produces traction that is used to wind the tether and generate power.

Tether Tension:

The tension in the tether is key, as the tether is what transfers the traction generated by the aircraft to the ground-based energy generation system. The tension in the tether increases during the descent

when the aircraft is subject to drag, and this enables the generator to produce power.

Gravity:

Gravity acts favorably during the descent, as it helps keep the aircraft moving downward, utilizing the force of traction without the need for external energy input.

Advantages of the "Eight" Mode

The 'eight' mode offers several advantages in AWE systems(Airborne Wind Energy):

Energy Efficiency: More efficient than traditional flight patterns, as the descent phase helps recover energy without requiring external input. Traditional flight patterns in AWE systems often include:

-Stationary or Hovering Mode: some systems rely on a hovering or stationary flight, similar to kites held in place by a tether. While this approach provides stability, it limits energy generation efficiency due to lower relative wind speeds.

-Circular Looping Pattern: in some designs, the airborne unit follows a continuous circular trajectory. While this allows for steady power generation, it often results in energy losses due to non-optimal aerodynamic forces during certain segments of the loop.

-Linear Yo-Yo Motion: another common pattern involves a periodic up-and-down movement, where the system ascends to generate power and then retracts with minimal resistance. This method can be effective but often requires additional energy input during the retraction phase.

By contrast, the 'eight' mode maximizes energy extraction by optimizing both the ascent and descent phases, leveraging aerodynamic lift and drag forces more efficiently.

Flight Stability: the continuous motion between ascending and descending phases enhances stability, reducing sudden fluctuations in traction and improving force management.

Power Optimization: the combination of lift and drag throughout the cycle allows for optimal power production. The descending phase generates power more steadily and continuously, improving system reliability.

Lower Energy Consumption: unlike conventional aircraft that require constant external energy to sustain upward flight, this mode enables a more energy-efficient flight cycle.

Challenges and Considerations

While the "eight" mode offers many advantages, there are some challenges and considerations:

Flight Control: Maintaining the aircraft along the correct trajectory requires precise control of the angle of attack, speed, and tether tension. Any errors in control can compromise the efficiency of the system.

Wind Variability: The production of energy is highly dependent on wind conditions. Turbulent or variable winds could negatively affect the trajectory and efficiency of the system.

Aircraft Design: The aircraft needs to be designed to withstand the forces of tension and drag during flight and to be stable enough to perform the "eight" cycle continuously.

2.6 Next steps

The next step is to automate the entire process, as the kite has the potential to remain airborne for several hours. Initially, both an automatic control model and the manual pilot will operate using the same parameters (such as sail positions, wind direction, etc.), and each will generate outputs that control the kite. The automatic model will function purely in a virtual environment for comparison purposes, allowing it to be assessed against the manual model. These outputs will be collected and stored for further analysis.

Once the data has been reviewed and a reliable, comparable performance between the two models has been established, the automatic system will be trusted to independently manage the entire process, with the joystick no longer required for regular operation, except in case of emergency.

CHAPTER 3

Mathematical model

The Flight Controller serves as a software designed to automate the control of the Kite, effectively replacing human intervention in steering the aircraft through a machine known as AWG(Airborne Wind Generator).

The forthcoming chapter will delve into the context within which the Flight Controller operates, including its interactions with the Kite, the machine, and other sensors. Subsequently, a detailed presentation of the Flight Controller will be provided, elucidating its components and functionality through block diagrams.

3.1 Context

The Flight Controller serves as a software that replaces human actions on the Pilots panel. Notably, the Automatic and manual modes are mutually exclusive. As illustrated in figure 3.1, the controller gathers measurements from the Weather station, AWG, and kite, utilizing this data to generate automatic inputs.

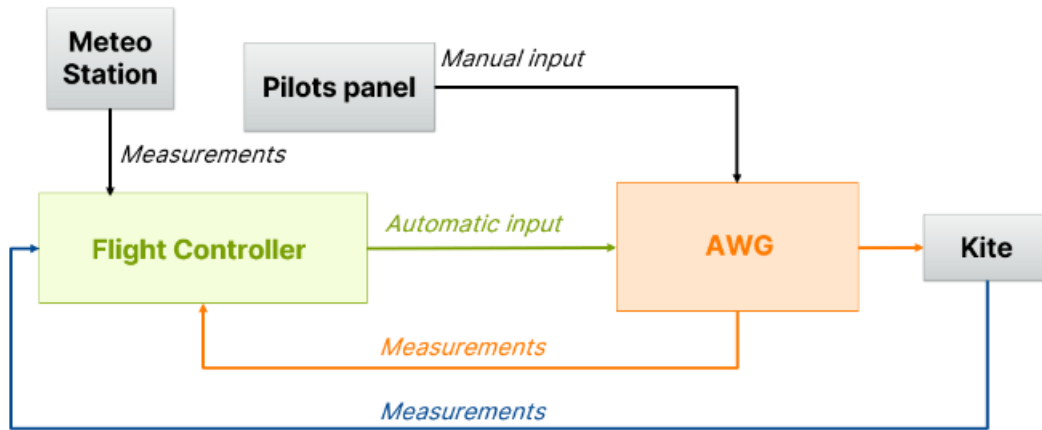


Figure 3.1: Block diagram system

In the upcoming section, the system architecture will be meticulously detailed, with a focused examination of the Flight Controller and AWG Blocks.

The Flight Controller operates with distinct commands, each tailored to the specific requirements of the four flight phases outlined below (see figure 3.2):

Generation (0): This phase marks the system's energy generation process.

Transition1 (1): Commencing after the maximum line length is reached, this phase initiates when the kite maneuvers towards the suitable position for the recovery phase.

Recovery (2): During this phase, the ropes are wound until the minimum cable length is attained.

Transition2 (3): As the minimum cable length is reached, the control mechanism acts to position the kite appropriately, preparing for the commencement of a new cycle with the generation phase.

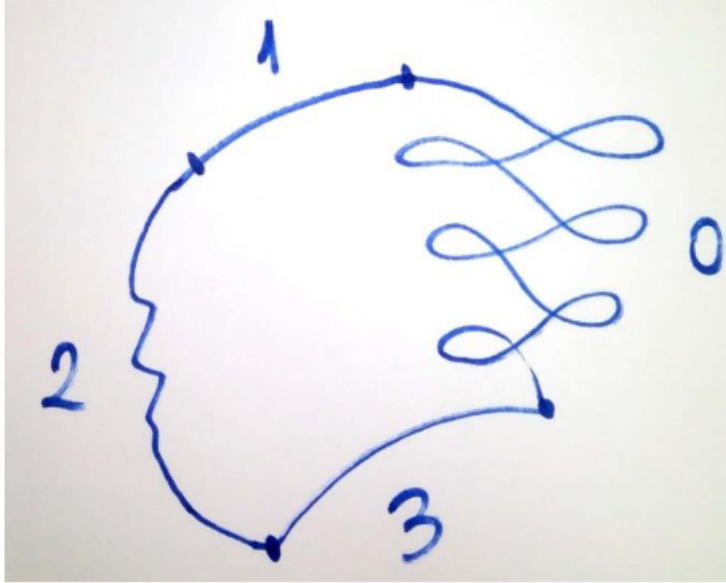


Figure 3.2: Trajectory Model

3.2 System architecture

The system is made up of two main parts that interact with each other, the Flight Controller and the AWG.

As illustrated in figure 3.3 the Flight Controller architecture is divided into four blocks:

- Trajectory control: is responsible for controlling the trajectory of the kite along the azimuth direction;
- Throttle Control: is responsible for controlling the kite entering and exiting;
- Supervisor: determines what the actual flight phase is;
- Conversion: transforms the kite measurements into a ground reference frame.

The AWG architecture can be divided into two blocks:

- Motion controller: it is the controller that receives signal references from both the Flight Controller and the Pilots panel and controls the electrical drives accordingly;
- Winches: are the assembly of electric motors, drums and electric drives.

The signals flowing between the blocks shown in Figure 2 are as follows: simotion(see Appendix for Simotion) signals in orange: • Lunwind: length of the unrolled laces of RED and GREEN winches (Lunwind-RED, Lunwind-GREEN);

- vwinch: speed of both red and green winches (vWinch-RED, vWinch-GREEN)

Flight controller signals in green:

- DL_{ref} : differential reference;
- $Throttle_{ref}$: Throttle reference.

Measurements from kite sensors in blue:

- position: position of the sail in the three geographical coordinates Latitude, Longitude,Altitude expressed globally reference framework;
- orientation: orientation of the sail expressed by the RLK matrix; it is a direct measurement from the avionics.

Measurements from kite sensors in black:

- AWG position is the AWG position expressed in Latitude-AWG, Longitude-AWG, Altitude-AWG, which will be set as a parameter once the AWG is entered into the field;
- The AWG orientation is the angle between the longitudinal axis of the AWG and named North Orientation-AWG ;
- WindDir is the measurement of the wind direction provided by the Weather Station.

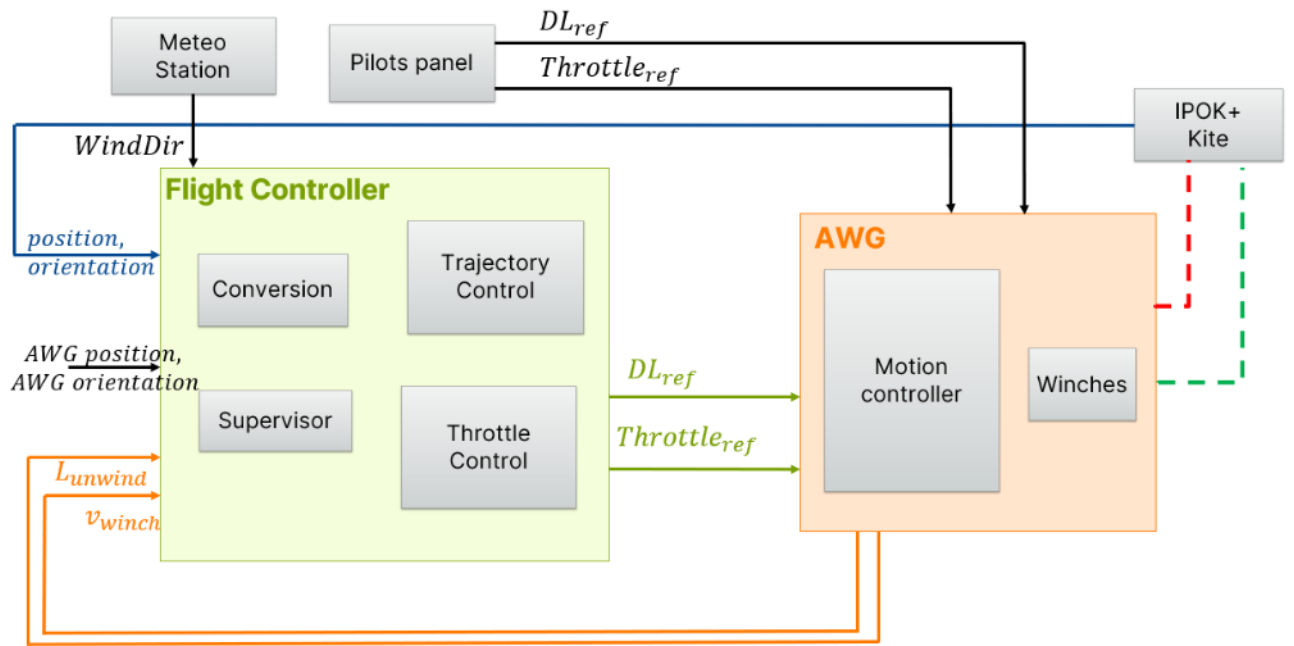


Figure 3.3: AWG Block Diagram

3.3 Reference frame

The model considers four reference frames:

- Global reference frame: its origin is at the point of Latitude 0, Longitude 0, Altitude 0 on the Earth's surface and is expressed in ENU (East-North-Up)(see Appendix for ENU);
- Ground reference frame: its origin is the center of AWG, using the axis referenced to the system itself;
- Kite reference frame: its origin is the center of the kite, using the axis of its body;
- Local zenith reference frame: it is a mobile reference frame on the surface of a sphere with a radius equal to the length of the rope; its origin is the center of the kite;
- Wind window reference frame: its origin is the center of AWG, and it moves according to the direction of the wind

3.4 Flight Controller components

Considering that the machine is operated by two separate controls, the flight controller was designed accordingly: the throttle and differential references are generated by two different control loops, Trajectory Control and Throttle Control, which are therefore decoupled.

The data flow is described below:

- flight phase: indicates what the actual flight phase is;
- phi-sign: indicates whether the azimuthal kite position is increasing or decreasing;
- (theta, thetadot): elevation kite position and relative time derivative in polar coordinates;
- (phi, phidot): azimuthal kite position and relative time derivative in polar coordinates;
- (range): velocity angle.

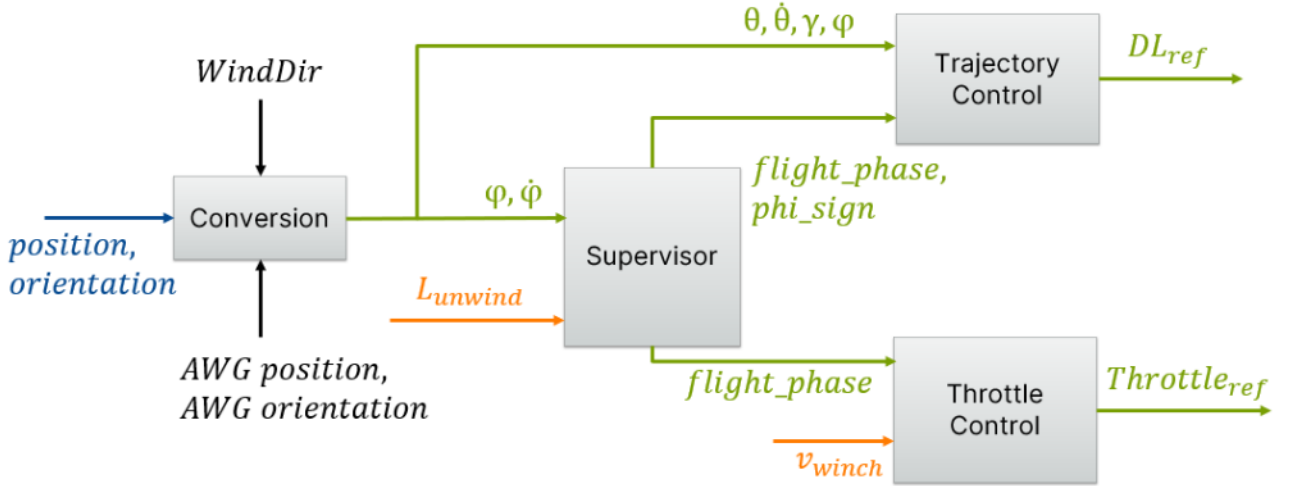


Figure 3.4: AWG Block Diagram

3.5 Reference frame

After obtaining position and velocity measurements from the onboard kite sensors, the flight controller identifies the correct flight phase to calculate the differential and throttle references resulting in common differential torque and torque references to the Simotion controller which will feed the AWG winches.

In Trajectory Control two controllers are responsible for generating the differential reference, switching each other depending on the flight phase.

3.6 Conversion

In order to create a fluid reading of the chapter the nomenclature is introduced:

R_{xy} is the rotation matrix that allows to rotate a vector from the Y reference system to the X reference system.

The variables obtained from IPOK are expressed in the global reference system.

IPOK is A GPS-equipped IMU (Inertial Measurement Unit) is a device that combines an inertial measurement unit (IMU) with a GPS (Global Positioning System) module.

The IMU (Inertial Measurement Unit) is a sensor that measures the acceleration, angular velocity, and direction of an object. It typically includes accelerometers and gyroscopes, which provide real-time data on position, speed, and orientation. The IMU is used to determine the movement of an object in three-dimensional space without the need for an external system.

GPS provides information about the geographic location of an object, allowing real-time determination of latitude, longitude, and altitude. When the IMU is combined with GPS, the system can use the GPS data to improve the accuracy of navigation data and compensate for errors that could arise from the IMU alone. This type of technology is used in advanced applications such as autonomous vehicle navigation, drones, military monitoring systems, and mobile device geolocation.

In this case, the IPOK is mounted on the kite to measure the data collected at high altitudes. This data will then be used as input for the algorithm that controls the entire system.

From now on, reference systems are expressed with the following sub-indexes:

- global reference system: L;
- ground reference system: G;
- Reference system for kites: K;
- local zenith reference system: Lz;
- wind window reference system: W.

The conversion block is made up of two main blocks:

- Position of the sail in the reference system of the wind window: the purpose of the block is to convert the coordinates of the Kite position (Latitude, Longitude, Altitude) to spherical ones (θ, ϕ, r) in a reference system that rotates with the direction of the wind.
- Gamma calculation: calculates the velocity angle (gamma).

RGL is the rotation matrix that transforms a vector from a global reference system to a ground reference system.

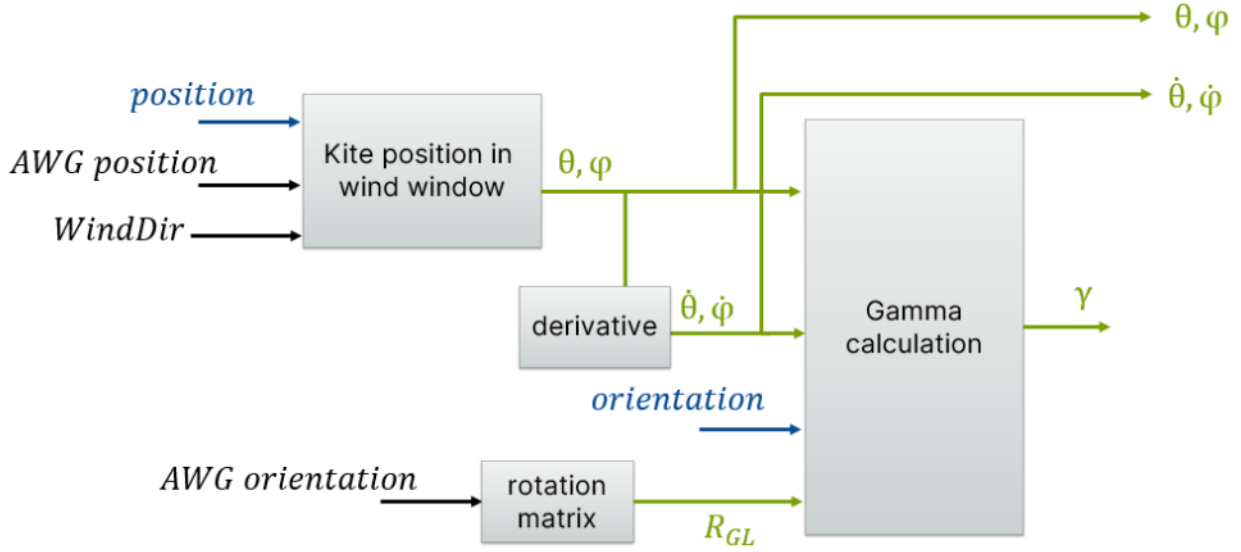


Figure 3.5: Conversion module

3.6.1 Kite position in the wind window

The following scheme describes calculation for positions with variables representing:

x_w, y_w, z_w is the kite position vector in cartesian coordinate w.r.t. wind window reference frame.

RWL is the matrix which rotates a vector from the global reference system to the wind window reference system.

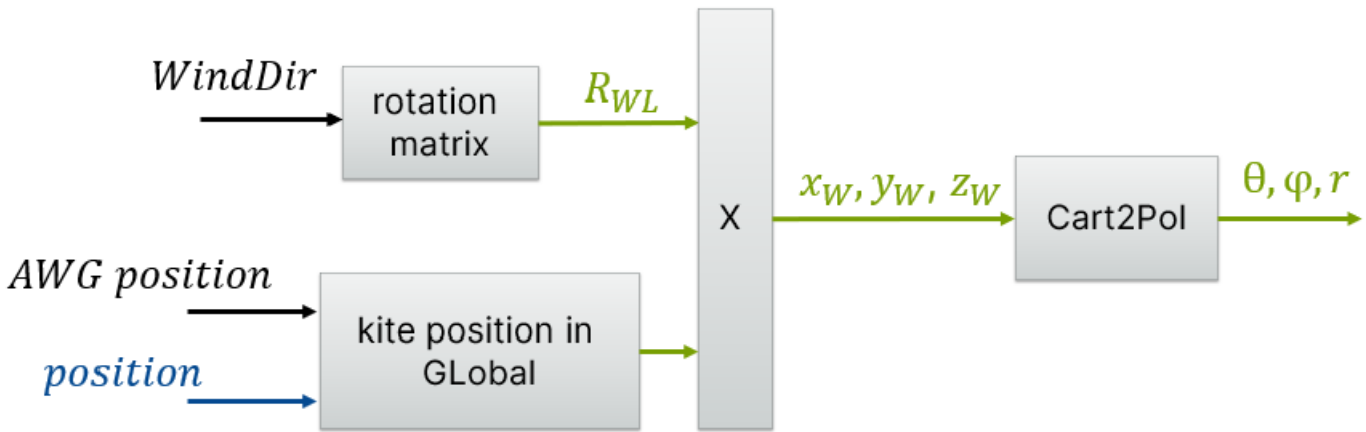


Figure 3.6: Kite position block

3.6.2 Rotation matrix

The RWL matrix is as follows:

$$R_{WL} = \begin{bmatrix} \sin(WindDir) & \cos(WindDir) & 0 \\ -\cos(WindDir) & \sin(WindDir) & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

Figure 3.7: Rwl matrix

Wind direction is expressed as the angle between the vector from which the wind comes and North. WindDir is considered clockwise positive. If the wind comes from S-W, the weather station reads 200°, to calculate WindDir:

$$WindDir = 200^\circ + 180^\circ = 1revolution(360^\circ) + 20^\circ$$

Figure 3.8: Wind direction formula

The maximum limit of the WindDir variable is 360°. If this maximum is exceeded it must be converted.

3.6.3 Position of the kite in global reference system

This block calculates the position of the kite relative to the position of the machine on the field (AWG), a difference between the relative positions is performed:

$$\begin{aligned} Dif_lon_m &= (Latitude - Latitude_AWG) * Degree2Meter_Lat; \\ Dif_lat_m &= (Longitude - Longitude_AWG) * Degree2Meter_Lon; \\ H_kite_rel_m &= Altitude - Altitude_AWG. \end{aligned}$$

Figure 3.9: Difference formulas

Where :

$$\text{Degree2Meter_Lat} = 111132.92 - 559.82 * \cos(2 * \text{Latitude_AWG}) + 1.175 * \cos(4 * \text{Latitude_AWG}) - 0.0023 * \cos(6 * \text{Latitude_AWG});$$

$$\text{Degree2Meter_Lon} = 111412.84 * \cos(\text{Latitude_AWG}) - 3.5 * \cos(3 * \text{Latitude_AWG}) + 0.1183 * \cos(5 * \text{Latitude_AWG});$$

Figure 3.10: Conversion factors

are the conversion factors from decimal degrees (DD) to meters for both latitude and longitude.

3.6.4 Multiplication

The position of the kite in the wind window reference system in Cartesian coordinates is calculated with the following formula:

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = [R_{WL}] \times \begin{bmatrix} \text{Dif_lon_m} \\ \text{Dif_lat_m} \\ \text{H_kite_rel_m} \end{bmatrix}_{ENU}$$

Figure 3.11: Kite position in wind window reference frame

3.6.5 Cart2Pol

This block performs the transformation of coordinates from Cartesian x_w, y_w, z_w to spherical coordinates θ, ϕ, r in the wind window reference system.

$$r = \sqrt{x_w^2 + y_w^2 + z_w^2};$$

$$\phi = \arctan\left(\frac{y_w}{x_w}\right);$$

$$\theta = \arctan\left[\frac{z_w}{\sqrt{x_w^2 + y_w^2}}\right].$$

Figure 3.12: Spherical conversions

3.6.6 Derivative

This block performs the derivative of θ, ϕ which are then used for the Gamma calculation. Furthermore, θ is used for generating reference theta while ϕ is also an input to the supervisor.

3.6.7 Rotation matrix

$$R_{GL} = \begin{bmatrix} \sin(\text{orientation_AWG}) & -\cos(\text{orientation_AWG}) & 0 \\ \cos(\text{orientation_AWG}) & \sin(\text{orientation_AWG}) & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

Figure 3.13: Rotation matrix

3.6.8 Gamma calculation

The velocity angle can be estimated by two methods. The `gamma-xAxisKite` variable will be calculated and used for the algorithm, while the `gammaFromPosition` variable will be calculated and saved for post-processing analysis

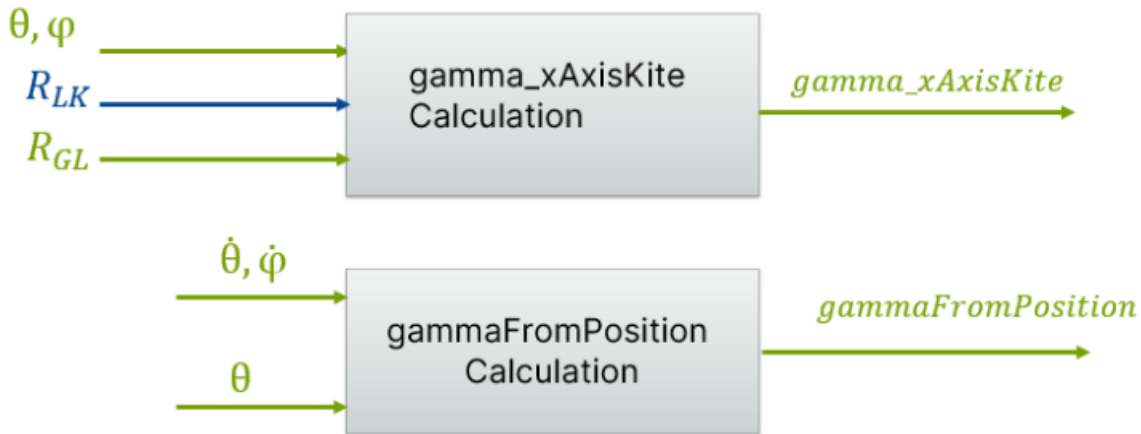


Figure 3.14: Gamma Calculation

Gamma calculation: `gammaxAxisKite`

This method takes into account both the kite position and orientation, the following formula is implemented:

$$\gamma = \arctan \left(\frac{xAxisKite_{Lz}(2)}{xAxisKite_{Lz}(1)} \right).$$

Figure 3.15: Gamma formula

Where `xAxisKiteLz` is a three-component vector obtained by considering the first row of the `RKLz` matrix.

To get that matrix you need to perform the following steps:

1. $R_{GK} = [R_{GL}]x[R_{LK}];$
2. $R_{LzK} = [R_{LzG}]x[R_{GK}]; \quad R_{KLz} = R_{LzK}';$
3. $xAxisKite_{Lz} \rightarrow \text{extract first row from } R_{KLz} \text{ matrix.}$

Figure 3.16: RLzG Matrix

RLzG is computed as follows:

$$R_{LzG} = \begin{bmatrix} -\cos(\varphi) \sin(\vartheta) & -\sin(\varphi) & -\cos(\varphi) \cos(\vartheta) \\ -\sin(\varphi) \sin(\vartheta) & \cos(\varphi) & -\sin(\varphi) \cos(\vartheta) \\ \cos(\vartheta) & 0 & -\sin(\vartheta) \end{bmatrix};$$

Figure 3.17: RLzG matrix

Gamma computation: `gammaFromPosition`

This method uses the derivative of the kite position in spherical coordinates according to the following formula:

$$\gamma = \arctan\left(\frac{\dot{\phi} \cos(\vartheta)}{\dot{\theta}}\right).$$

Figure 3.18: Gamma formula

The derivatives of θ and ϕ are calculated by numerical differentiation.

3.7 Supervisor

The supervisor is made up of two modules which deal with the detection of the flight phase and the calculation of the azimuth sign. These signals drive controller selection.



Figure 3.19: Supervisor module

Before performing the phase calculation, the arithmetic mean will be calculated between Lunwind-RED and Lunwind-GREEN

3.7.1 Flight phase detection

To identify the end of the generation and recovery phases, the condition choices are mandatory and are linked respectively to the maximum and minimum length of the ropes. While for the end of the two transition phases, the azimuth angle was taken into consideration for the switching conditions.

Starting from the Generation phase:

- If the actual cable length exceeds its maximum (MaxTetherLength), transition1 begins Transition1:
- when $\cos(\phi)$ maintains its value lower than CosPhiRec, the Recovery phase can begin. Recovery:
- When the tether length reaches its minimum value (MinTetherLength), transition2 is started Transition2:
- A new generation cycle is started when $\cos(\phi)$ exceeds the CosPhiGen threshold.

3.7.2 Sign computation

After calculating the flight phase and the derivative of phi, the sign calculation module extracts the information used to select the controller gain for the recovery phase. If the previous flight phase is transition 1 and the current flight phase is recovery, the following lines are executed:

$$\begin{aligned} & \text{if } \dot{\phi} > 0 \text{ then } \text{sign_phi} = -1 \\ & \text{else } \text{sign_phi} = 1 \end{aligned}$$

Figure 3.20: Sign computation

3.8 Trajectory Control

It is responsible for generating the differential reference based on different flight phases. Below controllers are described for each flight phase considering that:

- $DL_{ref,1}$ is generated by the gamma controller in the generation, transition1 and transition2 phases;
- $DL_{ref,2}$ is generated by the theta controller being fetched.

Only one differential reference is selected based on the current flight phase.

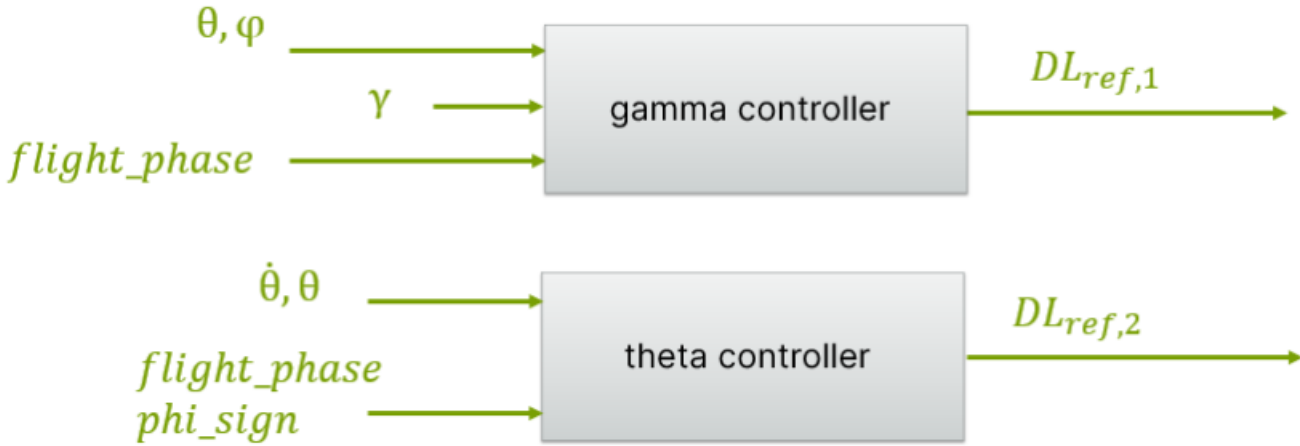


Figure 3.21: DL block diagram

3.8.1 Gamma controller

During the generation phase, the reference velocity angle γ_{ref} is calculated by switching the two target points to generate a figure of eight. Targets are selected based on the kite's instantaneous position in azimuth. A PI type controller works to reduce the error between reference range and

estimated range as shown in the figure below.

- γ_{ref} : speed angle reference;
- γ : estimated or calculated velocity angle (depends on the method used, as already explained previously).

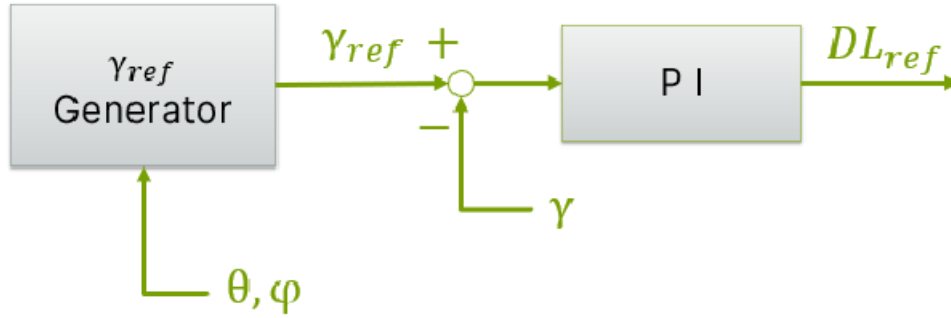


Figure 3.22: Gamma controller

Gammaref Generator

In the following image, the active target points are drawn.

Whenever the wing is close to the target points in terms of azimuthal position, the target point is moved to the other.

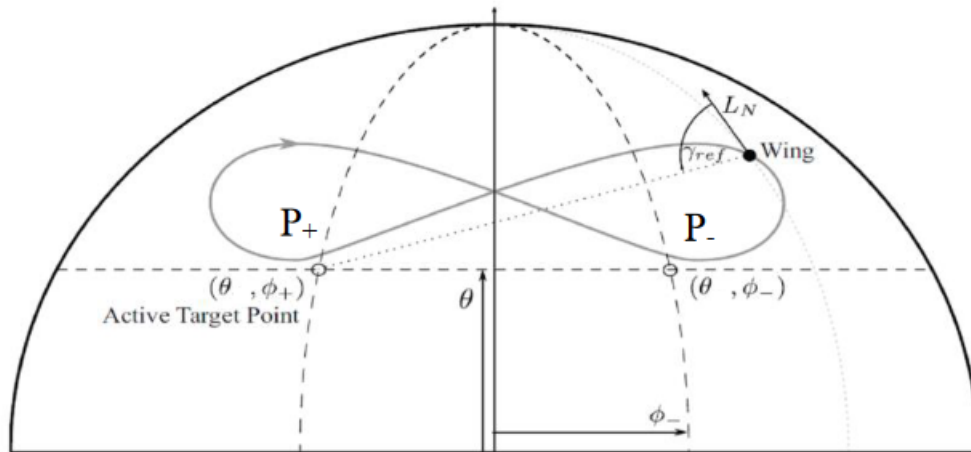


Figure 3.23: Kite motion range

The active target point is selected as indicated in the following strategy:

if $\varphi > \varphi_+$ *then* $P_a = P_-$
if $\varphi < \varphi_-$ *then* $P_a = P_+$
else $P_a = P_a$

where:

- $P_+ = (\varphi_+, \vartheta_t)(\text{phi_p}), P_- = (\varphi_-, \vartheta_t)$: target points;
 $\varphi_+, \vartheta_t \rightarrow \text{phi_p}, \text{theta_t}$
 $\varphi_-, \vartheta_t \rightarrow \text{phi_n}, \text{theta_t}$

- $P_a = (\varphi_a, \theta_a)$: active target point;
 $\varphi_a, \theta_a \rightarrow \text{phi_a}, \text{theta_a}$

Figure 3.24:

The velocity angle reference is calculated according to the following formula:

$$\gamma_{ref} = \arctan\left(\frac{(\varphi_a - \varphi)\cos(\vartheta)}{\vartheta_a - \vartheta}\right).$$

Figure 3.25: Active targets computation

To start the transition1 phase with the same velocity angle reference, the following formula is implemented if the current flight phase is transition1:

$$\gamma_{ref} = \text{sign}(\varphi_a) * \text{GammaRefTransition1}.$$

Figure 3.26: Velocity angle reference formula

PI controller

The PI controller calculates the differential length reference according to the following formula:

$$DL_{ref} = k_{P_{gamma}}(\gamma_{ref} - \gamma) + k_{I_{gamma}} \int (\gamma_{ref} - \gamma) + k_{D_{gamma}}(\dot{\gamma}_{ref} - \dot{\gamma}).$$

Figure 3.27: PID model

The equivalent discrete form is implemented using the following variables:

- GammaErr: difference between γ_{ref} and γ ;
- GammaSumErr: sum of GammaErr.
- Ts: sampling time.

By implementing the anti-windup strategy, a saturation value (DL-sat) is applied on the differential reference to be taken into consideration taking into account the mechanical limits of the AWG.

For the current implementation the PID derived part is equal to zero. Hence, $k_{D_{gamma}} = 0$ and derivative of reference gamma is not calculated.

3.8.2 Theta Controller

The differential reference is calculated according to the following scheme.

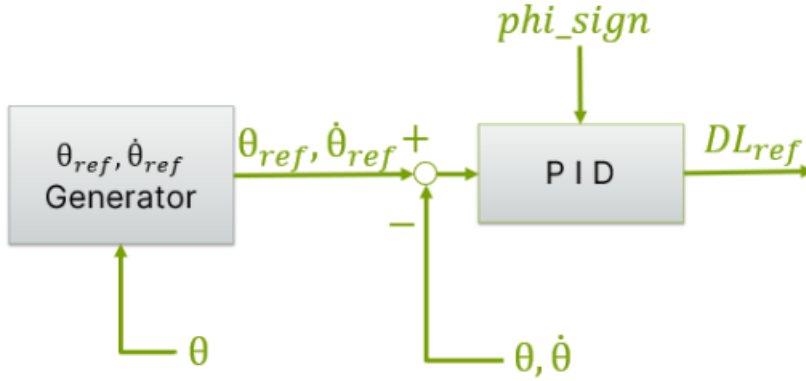


Figure 3.28: Theta Controller

A PID controller acts to reduce altitude error; The supervisor sign information changes the sign of the PID parameters. Theta reference is calculated in different ways depending on the flight phase.

Theta Ref Gen

The following formula is used to generate the theta reference (theta-ref), based on the Euler method for discrete signals:

$$\theta_{ref}(t) = \frac{\theta_{ref}(t-1) \tau + \theta_{SS} * T_s}{\tau + T_s}$$

Its time derivative (thetadot_ref) is calculated by numerical differentiation:

$$\dot{\theta}_{ref}(t) = \frac{\theta_{ref}(t-1) - \theta_{ref}(t)}{T_s}$$

Figure 3.29: Theta reference and its time derivative formulas

Where:

- T_s : sampling time;
- τ (tau): time constant of the first order transfer function used for generating reference Theta.
- θ_{SS} : steady-state value of the theta reference.

PID

The PID controller calculates the differential reference according to the following formula:

$$DL_{ref} = k_{P_{theta}}(\theta_{ref} - \theta) + k_{I_{theta}} \int (\theta_{ref} - \theta) + k_{D_{theta}}(\dot{\theta}_{ref} - \dot{\theta})$$

Figure 3.30: PID

If Φ sign is negative, the three PID parameters only change their sign. The equivalent discrete form is implemented using the following variables:

- ThetaSumErr: sum of ThetaErr.
- T_s : sampling time.

By implementing the anti-windup strategy, a saturation value (DL-sat) is applied on the differential reference to be taken into consideration taking into account the AWG mechanical limits.

3.9 Throttle Control

It deals with throttle reference generation by reducing the error by means of a proportional controller minimizing the error of speed and length of the winches. Use an empirical relationship between winch torque and speed, setting the following throttle reference to generation phase and negative winding speed. This formula is used when the system is on generation stage (step 0) and it required to control the throttle value.

$$Throttle_{ref} = \left(\frac{vWinch_{GREEN} + vWinch_{RED}}{2} \right)^2 * TorqueToVelRatio$$

Figure 3.31: Throttle reference

$$TorqueToVelRatio = \frac{\frac{1}{2} * \rho * S * C_L * E^2 * \sqrt{\left(1 + \frac{1}{E^2}\right)^3} * CorrTtoV}{a_ratio}$$

Figure 3.32: Torque to velocity ratio conversion

Where:

- rho: air density at standard conditions;
- S: surface of the kite;
- CL: lift coefficient;
- E: kite efficiency;
- CorrTtoV: experimental correction factor for the torque/speed ratio of the square winch.
- a-ratio: conversion factor from angular position to linear position (can be approximated with the inverse of the winch radius).

Meanwhile for the other three phases, the throttle reference is set to a constant value.

CHAPTER 4

Graphics

The subsequent phase involves implementing graphics.

These graphics hold significant importance, serving to visualize the trajectory of the model onscreen, originating from the kite's coordinates.

Moreover, during the completion of prototyping and the initiation of field tests, graphics play an important role. A comparative analysis between manual and automatic piloting becomes feasible. The automatic pilot model conducts simulations with identical data to the manual model, displaying the kite's relative position onscreen during flight. This facilitates a direct visual comparison between the two models(ON-FIELD analysis).

4.1 Cubical trajectory

For the graphics component, the decision was made to employ UNITY software, a versatile graphics engine developed by Unity Technologies.

Unity is a powerful and widely used game development platform that allows developers to create both 2D and 3D games and interactive experiences.

It provides a comprehensive suite of tools for designing, programming, and deploying applications across various platforms, including PC, mobile devices, consoles, and virtual/augmented reality systems. Unity's user-friendly interface enables artists and developers to collaborate efficiently, combining creative assets with advanced coding capabilities. The platform includes features like physics simulation, real-time rendering, and asset management, making it a versatile tool for game design, animation, and interactive media development. With a strong emphasis on ease of use, flexibility, and cross-platform compatibility, Unity is one of the leading engines for game development and real-time graphics.

The initial step involved acquiring proficiency in using this graphic software. As a practice exercise, the task was set to manipulate an object along a cubic trajectory. At each iteration, the object receives a set of coordinates (X, Y, Z) and is required to move towards the new position from its current location.

To achieve this, a script written in the CSharp language was implemented in UNITY. UNITY scripts bear similarities to those of ARDUINO, featuring a `START()` section for initializing parameters and an `UPDATE()` area for program updates at each frame. Initially, three position arrays (X, Y, Z) were defined to designate the target position of the mobile object. Each set represents the coordinates of an edge of a cube.

Following the establishment of the object's initial position in the `START()` section, the object orients itself toward a new position by extracting coordinates from the position arrays. Subsequently, the object moves towards this updated target using the `MoveTowards` function.

With each iteration, the target positions undergo updates, facilitating the object's movement along a cubic trajectory. Upon reaching the last triad, the new target becomes the set located at position '0' ensuring the continuous movement of the cube as it follows a cubic trajectory.

Nevertheless, the current script encounters an issue as the `MoveTowards` function necessitates multiple frames to complete its operation. Consequently, in each iteration, the target is updated before the function has finished its execution.

To address this challenge, a new condition has been incorporated. This condition ensures that the target is not updated until the `MoveTowards` function has successfully concluded its operation.

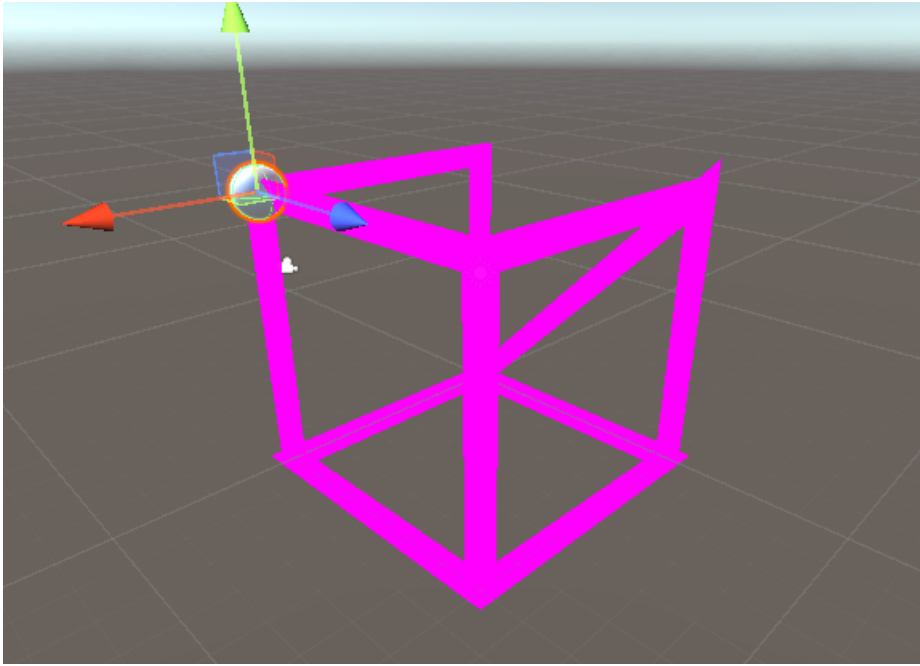


Figure 4.1: Cubic trajectory example

4.2 Graphics server/client implementation

The next step is to recreate the previous exercise but the target coordinates are sent from an external program.

In particular, the external program is a Client, written in C++, which communicates with UNITY via TCP/IP protocol.

So at UNITY script level a SERVER will be implemented that will receive the data and use it to recreate the cubic movement. The initial task is to implement the Client in C++. The Client is required, at each iteration, to transmit the coordinates of the Target via TCP/IP. The package carrying information regarding the new point utilizes a JSON structure. This ensures that each field is accompanied by its corresponding value, eliminating any potential ambiguity.

During each iteration, the Client loads the coordinates of the new target onto an object of class type named "st-Coord." This approach was chosen to leverage the capabilities of the Json.cpp library. Specifically, within this library, there exists a function that facilitates the conversion of an object from Class type to Json type.

In preparation for transmission via TCP, the packet undergoes a transformation into a string. Serialization is executed using the JSON.cpp library, enabling the transition from a JSON object to a string. Subsequently, the Server receives this string and is tasked with performing deserialization,

retrieving the JSON structure from the string.

Upon implementing the Client, and in the absence of the Server, a testing procedure is desired. To facilitate this, a TCP Server is being created using Hercules.

Upon configuring the IP and Port for the connection, the strings sent by the C++ Client could be observed on the screen.

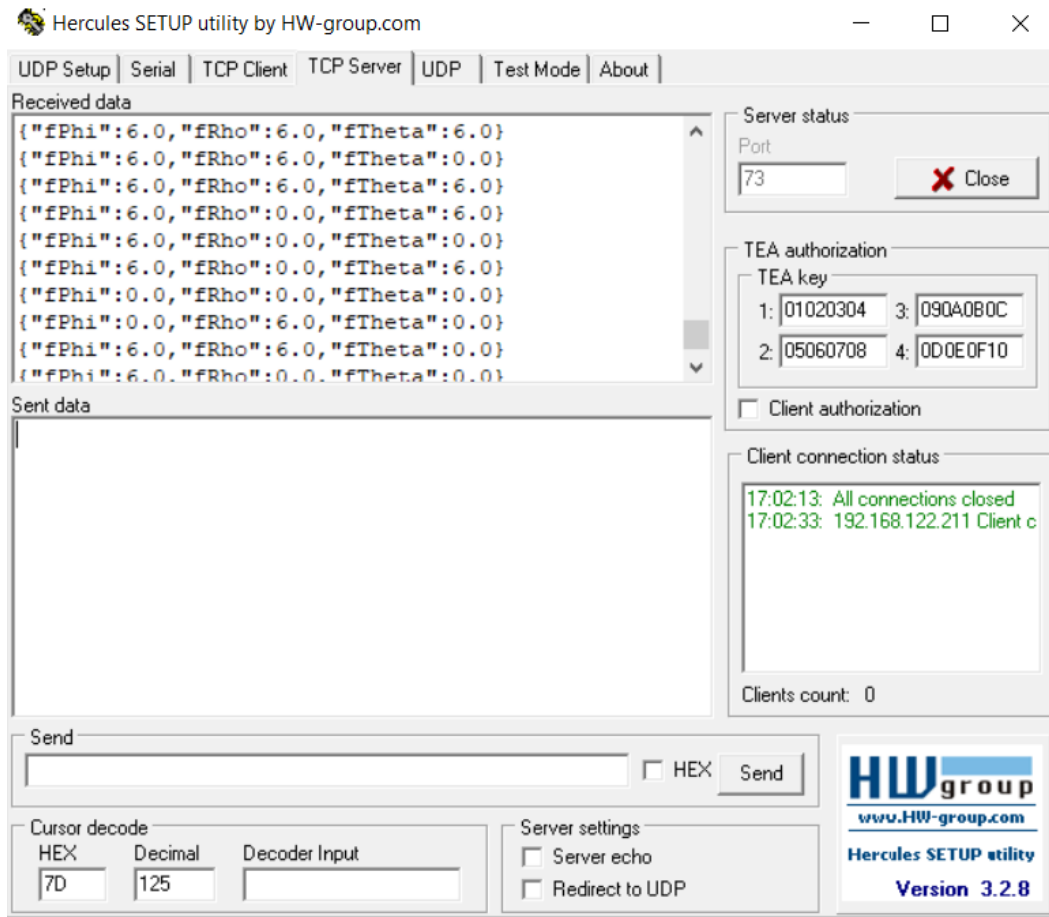


Figure 4.2: Hercules software

Note the Json-like structure of each string, where each field is accompanied by its corresponding value. The subsequent step involves implementing the Server in Unity. This requires the creation of a script designed to receive coordinates from the Client and update the new target. Consequently, this approach facilitates the transmission of coordinates between computers, enabling the realization of a cubic trajectory, similar to the previous exercise.

4.3 Trajectory implementation with spherical trajectories

The subsequent task involves modifying the movement of the object to enable it to follow a spherical trajectory, mirroring the '8' motion of the kite.

Initially, the intention was to maintain the same technique employed in the previous exercise: providing a set of coordinates for the object to orient towards.

However, this approach proves impractical for replicating a wave-like movement, as it requires a large number of coordinates.

To overcome this challenge, a decision was made to adopt a different method: changing the coordinates through keyboard commands.

Specifically, the modification involves pressing Q to increase the radius of a unit and pressing A to decrease its value. Likewise, pressing W or S increases or decreases Phi by 5°, and Theta is adjusted with E and D.

Following the initialization of spherical coordinates Rho, Theta, and Phi, their values can be dynamically altered through these keyboard commands. This process effectively creates a new target for the object to aim at.

Parallel to the method employed in the prior exercise, the coordinates are stored in the Json class and transmitted to the CSharp Server. However, a notable challenge arises: Unity operates solely within a Cartesian reference system, not a spherical one. Consequently, each spherical triple must be converted into Cartesian coordinates using trigonometric formulas.:

$$x = r \sin(\phi) \cos(\theta)$$

$$y = r \sin(\phi) \sin(\theta)$$

$$z = r \cos(\phi)$$

Figure 4.3: Trigonometric formulas

However, there must be adjustments since Unity's reference system is y-up, i.e. Z and y are reversed compared to the standard reference system.

After making these adjustments, it is possible, by typing the keys, to see the object move along a spherical trajectory.

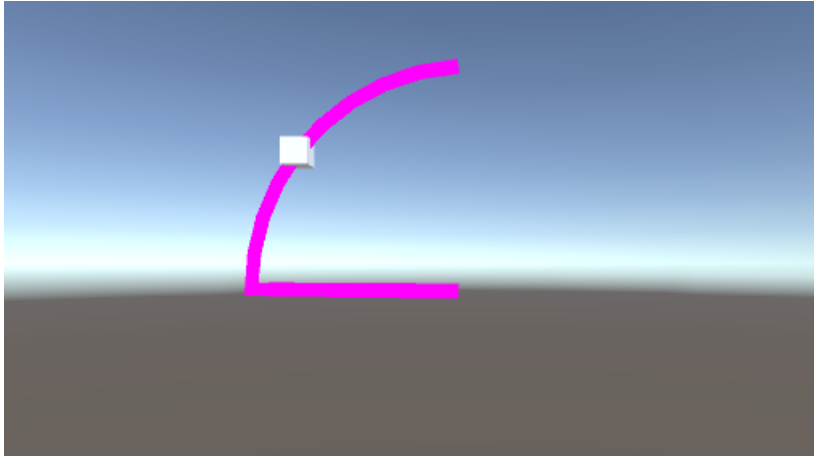


Figure 4.4: Spherical movement example

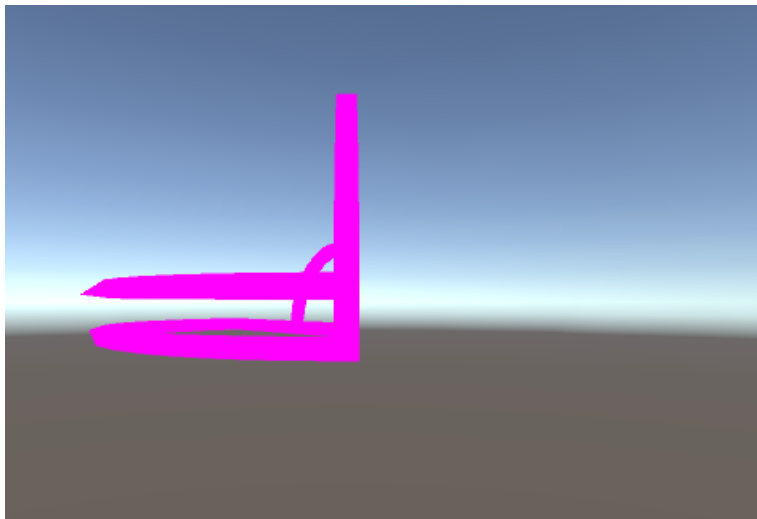


Figure 4.5: Spherical movement from a transverse camera

4.4 Simulation trajectory visualization

The next goal is to visualize the complete movement of the kite. To achieve this, the spherical coordinates were extracted from the Matlab/Simulink model.

The file generated by Matlab is quite large, prompting the decision to extract one row every N rows.

The Matlab model samples at a very high frequency, and consecutive rows have minimal differences. Therefore, a C++ program was developed to read the file created by Matlab and save one line every N lines in another file. This processed file will then be utilized as input for the C++ Client.

The program responsible for generating the file has been kept separate from the C++ Client, primarily due to concerns about potential performance issues when working with a large file (approximately 180k lines).

In this setup, the C++ Client reads the file produced by the external program. Each line in the file contains spherical coordinates of the new target (Rho, Theta, Phi). These coordinates are then transmitted to the Unity Server, which processes them to guide the object along the trajectory described in the file. This approach helps manage the computational load and ensures smoother performance.

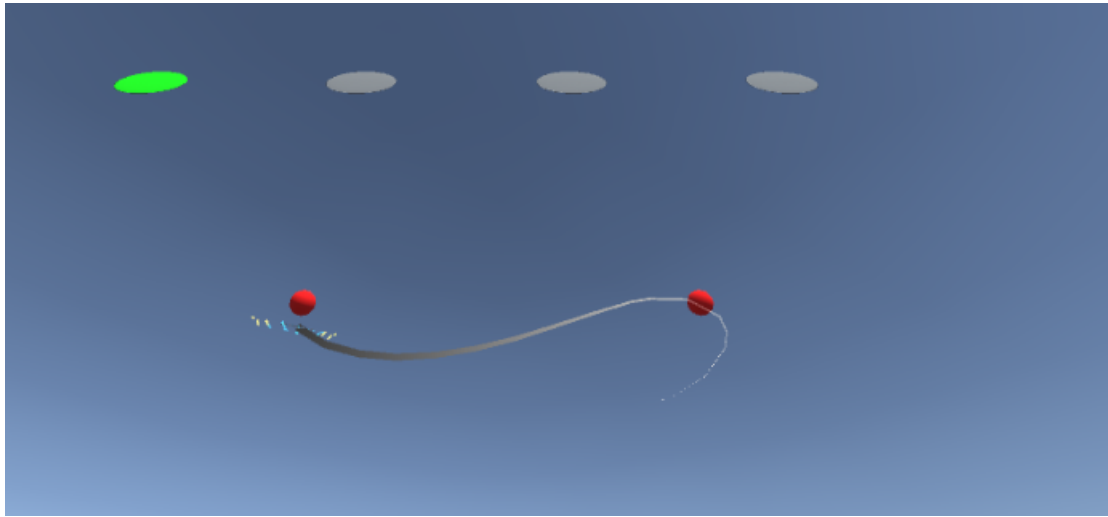


Figure 4.6: "Eight" movement

The next objective is to optimize the entire process, aiming for automation. Currently, to initiate the Unity program, it's needed to click on the Play button, wait for it to load, and then run the C++ Client.

To streamline this, the initial approach was to extract the Unity .exe and activate it during the Client's initialization.

However, this strategy faces challenges, as the call to the Unity .exe requires continuous use of the C++ Console, impeding the seamless continuation of execution. The Console remains engaged in communication with Unity, preventing the program's smooth progression.

To address this challenge, two strategies were identified: the first involves the use of MultiConsole at

the C++ level (utilizing one Console for calling the Unity .exe and another for programming related to the Client), while the second idea is to have Unity call the C++ Client executable.

After a thorough evaluation of both solutions, the decision was made to proceed with the second strategy. The first strategy was deemed complex to implement, and the second strategy appeared to offer a more straightforward and feasible approach.

To provide an efficient solution, a button was added to the Unity interface, which, when pressed, triggers the execution of the C++ Client executable.



Figure 4.7: List of buttons in order to start/stop the executable or for changing the camera

4.5 Camera

A critical aspect of the graphics is the choice of the observation point, determining the angle from which the movement of the kite is viewed.

To facilitate this, a cubic object named AWG has been introduced at the origin (0,0,0). This object is intended to represent the turret of the kite on the ground, serving the purpose of showcasing the movement of the sail around this central point. This addition contributes to providing a clear and illustrative perspective on the kite's motion.

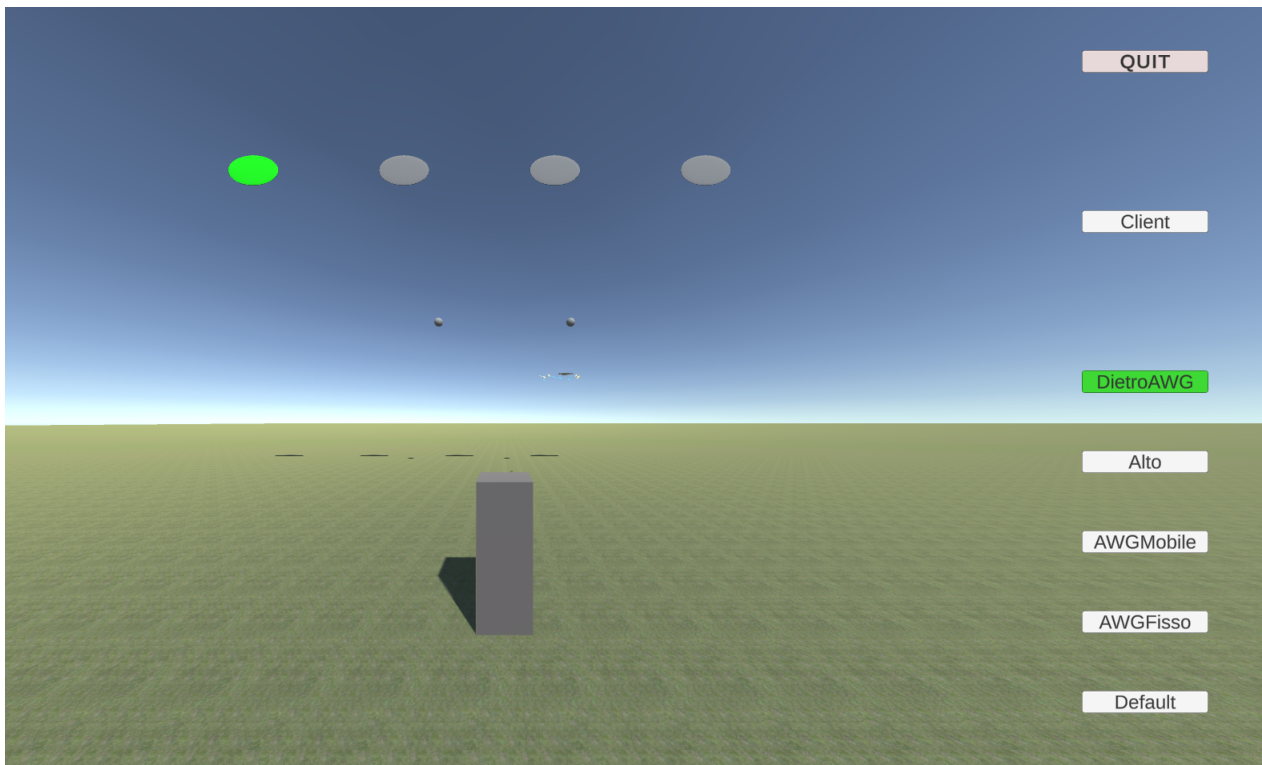


Figure 4.8: Behind AWG camera

Consequently, the camera was positioned behind the AWG to achieve a comprehensive visualization of the kite's motion. However, this perspective can present challenges in understanding certain phases of the flight. For instance, during the Transition1 phase, where the glider moves laterally, it might appear as if the glider continues to rise when viewed from behind the AWG. To address this display issue, an alternative is to change the observation point, allowing the movement of the sail to be viewed from above.

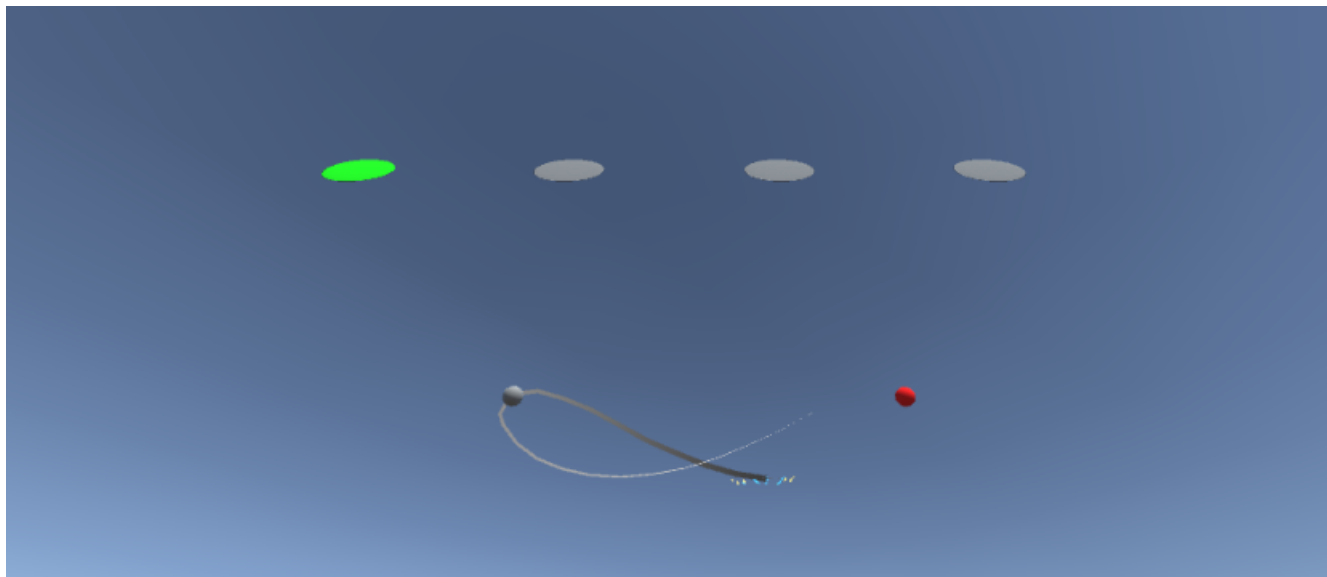


Figure 4.9: AWG point of view camera

By adopting this overhead perspective, it becomes apparent how the target points are updated. Specifically, during these updates, Θ remains constant while only Φ varies, providing a clearer insight into the dynamics of the kite's motion.

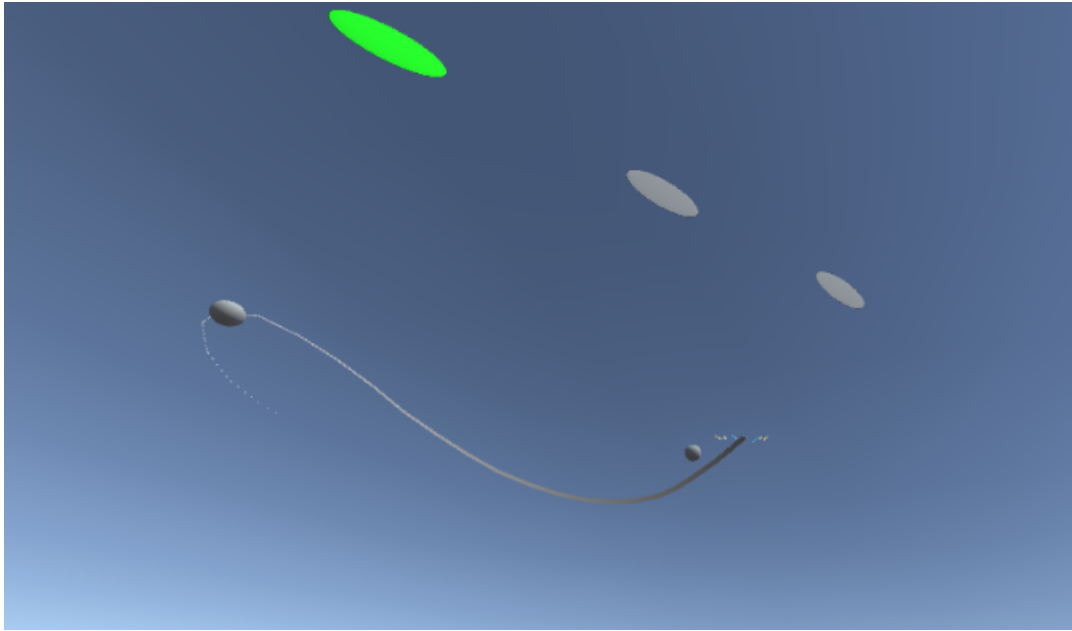
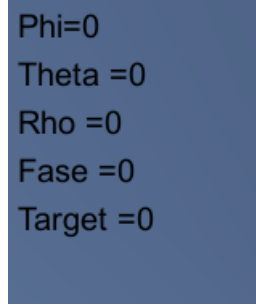


Figure 4.10: "Chasing" kite camera

However, this observation point fails to clearly display the 'figure-eight' movement of the sail. In response, a model was devised wherein the view can be dynamically chosen by modifying the camera settings. Consequently, buttons were added to the screen, allowing users to switch between different views and achieve a more optimal visual representation.

Three buttons were introduced with the following labels: 'MAIN', 'AWG', and 'TOP'. The 'Main' camera corresponds to the view behind the AWG, 'TOP' offers a top-down perspective, and 'AWG' aligns the camera with the turret, providing a first-person view as if situated on the AWG itself. This approach allows users to select the most suitable perspective for a clearer and more comprehensive understanding of the kite's motion.

Indeed, by incorporating these camera options and allowing users to switch between different views, a comprehensive and detailed visualization of the kite's movement is achieved. This approach provides a flexible and user-friendly interface, enabling a better understanding of the dynamic 'figure-eight' motion of the kite from various perspectives.



Phi=0
Theta =0
Rho =0
Fase =0
Target =0

Figure 4.11: in the moment it's displayed the actual position and the actual phase

CHAPTER 5

Hardware

During the design phase, careful considerations were taken into account, particularly at the hardware level and, crucially, at the operating system level. The primary objective was to proactively address potential issues related to the performance of the entire process. This strategic approach aimed to ensure the robustness and efficiency of the system by anticipating and mitigating potential challenges in both hardware and operating system environments.

5.1 PC WINDOWS general purpose

Initially, the plan was to utilize a general-purpose computer supporting the Windows operating system for implementing the Flight Controller. However, during the testing phase, challenges emerged related to the cyclical nature of the Windows operating system. Specifically, Windows does not sample constantly but depends on the state of the CPU.

To illustrate this behavior, a C++ program was implemented to demonstrate the non-linearity of the Windows operating system. The full system operates on the order of thousands of hertz. Therefore, a program was created to test the behavior of Windows under similar conditions as the system. The test involves printing a message on the screen for every millisecond that passes. In cases where the update print exceeds 1 millisecond, an error message is displayed on the screen.

Initially, the test was conducted when the processor is relatively idle (no background programs running). In this scenario, the program successfully prints the message every millisecond without errors. However, when the CPU is busy, the cyclical nature of the print message is disrupted, and the program prints the message along with its relative delay.

```

C:\Users\farah\OneDrive\Desktop\workspace\TestWhileTime\Debug\TestWhileTime.exe
Timestamp: 2025-01-08 15:14:11.4440003
Timestamp: 2025-01-08 15:14:11.4690003
Timestamp: 2025-01-08 15:14:11.4940003
Timestamp: 2025-01-08 15:14:11.5190002
Timestamp: 2025-01-08 15:14:11.5440477
Timestamp: 2025-01-08 15:14:11.5690003
Timestamp: 2025-01-08 15:14:11.5940002
Timestamp: 2025-01-08 15:14:11.6190004
Timestamp: 2025-01-08 15:14:11.6440002
Timestamp: 2025-01-08 15:14:11.6690003
Timestamp: 2025-01-08 15:14:11.6940001
Timestamp: 2025-01-08 15:14:11.7190002
Timestamp: 2025-01-08 15:14:11.7440001
Timestamp: 2025-01-08 15:14:11.7690002
Timestamp: 2025-01-08 15:14:11.7940003
Timestamp: 2025-01-08 15:14:11.8190003
Timestamp: 2025-01-08 15:14:11.8440003
Timestamp: 2025-01-08 15:14:11.8690003
Timestamp: 2025-01-08 15:14:11.8945102
Timestamp: 2025-01-08 15:14:11.9380618
Timestamp: 2025-01-08 15:14:11.9800004
Timestamp: 2025-01-08 15:14:12.0050003
Timestamp: 2025-01-08 15:14:12.0300003
Timestamp: 2025-01-08 15:14:12.0550003
Timestamp: 2025-01-08 15:14:12.0800005
Timestamp: 2025-01-08 15:14:12.1050005
Timestamp: 2025-01-08 15:14:12.1300005
Timestamp: 2025-01-08 15:14:12.1550003
Timestamp: 2025-01-08 15:14:12.1800004
Timestamp:

```

Figure 5.1: Case when the CPU is "working" at low rate

```

Timestamp: 2025-01-08 15:15:33.9732368
Timestamp: 2025-01-08 15:15:34.0110003
Timestamp: 2025-01-08 15:15:34.2040004
Timestamp: 2025-01-08 15:15:34.3233473 ERRORE sul ciclo di 6 millisecondi!
Timestamp: 2025-01-08 15:15:34.4060003
Timestamp: 2025-01-08 15:15:34.4460004
Timestamp: 2025-01-08 15:15:34.5167701 ERRORE sul ciclo di 13 millisecondi!
Timestamp: 2025-01-08 15:15:34.6720004
Timestamp: 2025-01-08 15:15:34.7509174 ERRORE sul ciclo di 4 millisecondi!
Timestamp: 2025-01-08 15:15:34.9030004
Timestamp: 2025-01-08 15:15:34.9310005
Timestamp: 2025-01-08 15:15:34.9570003
Timestamp: 2025-01-08 15:15:34.9850006
Timestamp: 2025-01-08 15:15:35.0120003
Timestamp: 2025-01-08 15:15:35.0380021
Timestamp: 2025-01-08 15:15:35.0640005
Timestamp: 2025-01-08 15:15:35.0890004
Timestamp: 2025-01-08 15:15:35.1166117 ERRORE sul ciclo di 2 millisecondi!
Timestamp: 2025-01-08 15:15:35.1540008
Timestamp: 2025-01-08 15:15:35.1790032
Timestamp: 2025-01-08 15:15:35.2050106
Timestamp: 2025-01-08 15:15:35.2631723 ERRORE sul ciclo di 9 millisecondi!
Timestamp: 2025-01-08 15:15:35.3280006
Timestamp: 2025-01-08 15:15:35.3850007
Timestamp: 2025-01-08 15:15:35.4410008
Timestamp: 2025-01-08 15:15:35.4696210

```

Figure 5.2: Case when Matlab is "starting working"

Addressing the issue is imperative as the Flight Controller algorithm relies on PID-type controllers, which are highly sensitive to jitter. The presence of jitter can lead to partially incorrect results, causing the Glider to execute inaccurate commands.

To mitigate this problem, the decision was made to opt for a Siemens IPC industrial computer. This system comprises a SoftPLC in addition to the Windows operating system. This choice aims to provide a more stable and reliable environment, especially for precision-driven PID controllers, ensuring that the Flight Controller can operate with minimal jitter and deliver accurate commands to the Kite.

5.2 IPC Siemens

Siemens IPC (Industrial PC) and SoftPLC (Software PLC) are integral components in industrial automation systems, combining hardware and software solutions for efficient control and monitoring of machinery and processes. Here's an explanation of their features:

Siemens Industrial PCs (IPC) are ruggedized, high-performance computers designed for industrial environments. They provide computing power for complex tasks, such as data collection, control, and communication in factories and manufacturing plants. Here are the key features:

Robust Design:

Siemens IPCs are built to withstand harsh industrial environments, including extreme temperatures, dust, vibrations, and electromagnetic interference. They are certified for use in industrial settings, meeting various standards such as CE, UL, and others.

High Performance:

Siemens IPCs are equipped with powerful processors (e.g., Intel Core i-series) to handle demanding tasks like data processing, machine control, and real-time analysis.

The performance makes them suitable for handling complex operations and running advanced software applications.

Flexibility:

Siemens offers different IPC models, such as panel PCs, rack-mounted PCs, and embedded systems, to suit various industrial use cases.

They are highly customizable with flexible I/O options, expansion slots, and connectivity ports, allowing integration with other systems.

Real-time Capabilities:

Some Siemens IPCs support real-time operating systems (RTOS), providing precise control and monitoring of industrial processes in real time.

This is critical for applications where delays cannot be tolerated, such as in machine automation or

safety systems.

Communication and Connectivity:

Siemens IPCs come with extensive connectivity options, including Ethernet, USB, serial ports, and industry-specific protocols such as PROFINET, PROFIBUS, and Modbus.

They support seamless integration with PLCs, SCADA systems, and other automation devices.

Industrial Software Compatibility:

Siemens IPCs are compatible with various industrial software tools, including Siemens' own TIA Portal, WinCC for HMI (Human-Machine Interface), and SIMATIC SCADA.

This makes them a good fit for Siemens-based automation systems.

Long Lifecycle and Support:

They are designed for long-term use in industrial settings, with a long product lifecycle and support for software updates and maintenance.

SoftPLC (Software PLC)

A SoftPLC is a software-based implementation of a traditional hardware PLC (Programmable Logic Controller). It runs on industrial PCs or standard computers instead of specialized hardware. Here are its key features:

-Software-based Control:

SoftPLC enables PLC functionality to be performed by software running on general-purpose computers (like Siemens IPCs).

It eliminates the need for dedicated hardware PLC devices, providing cost-effective and flexible control solutions.

-Flexibility and Scalability:

SoftPLC systems are highly flexible and can be scaled to suit different levels of automation, from small systems to large, complex networks.

They can integrate with various industrial control devices, such as sensors, actuators, and robotics, and can be adapted to different types of applications.

-Real-time Control:

Like traditional PLCs, SoftPLC systems support real-time control and deterministic operation, ensuring that the control logic runs without delay. They can handle tasks such as machine control, process monitoring, and data logging.

-High-level Programming Languages: SoftPLCs support high-level programming languages such as IEC 61131-3, which includes Ladder Logic (LD), Function Block Diagram (FBD), Structured Text (ST), Instruction List (IL), and Sequential Function Chart (SFC). This allows for easier programming

and modification of control logic compared to traditional hardware PLCs.

-Cost-Effective:

SoftPLC offers significant cost savings by using standard hardware, such as industrial PCs or servers, instead of dedicated PLC hardware.

It also reduces the need for specialized control hardware when scaling the system.

-Integration and Communication:

SoftPLC systems can easily integrate with other control and automation systems, as they support a wide range of communication protocols like OPC, Modbus, PROFINET, and Ethernet/IP.

This makes them ideal for distributed control systems, remote monitoring, and communication with SCADA or HMI systems.

-Advanced Features:

SoftPLCs often come with advanced features such as data analytics, remote diagnostics, and system monitoring tools.

They can be combined with other software applications for enhanced data processing, machine learning, and predictive maintenance.

-Continuous Updates and Customization:

Unlike traditional PLCs, which have fixed hardware configurations, SoftPLC systems are easily updated and customized to meet changing needs and new standards.

-Compatibility with Industry Standards:

Siemens' SoftPLC solutions are fully compatible with international standards, such as IEC 61131-3, and Siemens' own protocols, including PROFINET and PROFIBUS.

Siemens IPC and SoftPLC Integration

When Siemens IPCs are used in conjunction with SoftPLC systems, they provide a powerful, flexible, and scalable solution for industrial automation. The IPC serves as the hardware platform for running the SoftPLC software, which in turn provides control and automation capabilities.

Key Benefits of Siemens IPC and SoftPLC:

Reduced hardware costs: By using a SoftPLC on a Siemens IPC, you eliminate the need for dedicated PLC hardware.

Increased flexibility: Both Siemens IPCs and SoftPLCs offer flexibility in terms of programming, scaling, and integration with other systems.

Enhanced performance: Siemens IPCs offer the processing power and real-time capabilities necessary for running demanding SoftPLC applications.

Easy integration: Siemens SoftPLC solutions integrate seamlessly with other Siemens devices and

third-party systems, allowing for comprehensive automation solutions.

In summary, Siemens IPCs and SoftPLCs provide an efficient, scalable, and flexible approach to industrial control systems, enabling companies to reduce costs, improve performance, and enhance automation capabilities.

Therefore, using the Siemens IPC there is no problem of the cyclicity of the entire system regardless of the state of the CPU, therefore each process is deterministic.

5.3 HARDWARE and Interconnections

The system is composed of:

PC SIEMENS IPC 427E: receives the data relating to the position of the sail and the direction of the wind and processes the algorithm to generate the rope length differential and the reference torque to send to the Simotion.

SIMOTION: receives the signals useful for piloting the Kite from the PC

IPOK: Provides data relating to the speed and position of the Kite

WEATHER STATION: provides data relating to weather conditions



Figure 5.3: Block Diagram

The Siemens PC is composed (as previously described) of a Windows operating system and a CPU 1507S softPLC system.

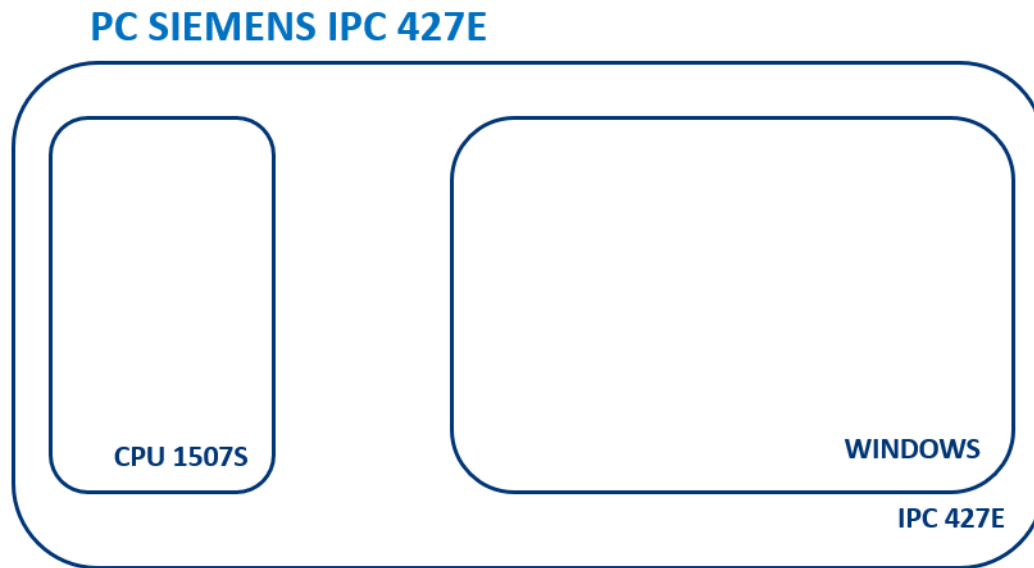


Figure 5.4: PC Siemens

The softPLC system offers the possibility of developing applications in advanced languages and integrating them into the controller. ODK is a development package and serves as an interface for calling proprietary programs in advanced languages from the CPU controller.

Therefore the ODK allows the use of function blocks (FB), which will be implemented to develop the Flight Controller algorithm.

C++ code must be compiled as a dynamic-link-library (DLL), which helps promote code modularization, code reuse, efficient memory use, and disk space reduction. So, the operating system and programs load faster, run faster, and require less disk space on the computer.

5.3.1 DLL

Several DLLs have been implemented, each of which will have a well-defined role:

Ipok Data DLL

DLL Weather station data

DLL Conversion

DLL Algorithm

DLL Client Coordinates

Ipok Data DLL: this function block is used to receive data relating to the coordinates, position and

speed of the sail from the IPOK. Communication with the IPOK occurs via TCP/IP communication by implementing the TCP server in C++.

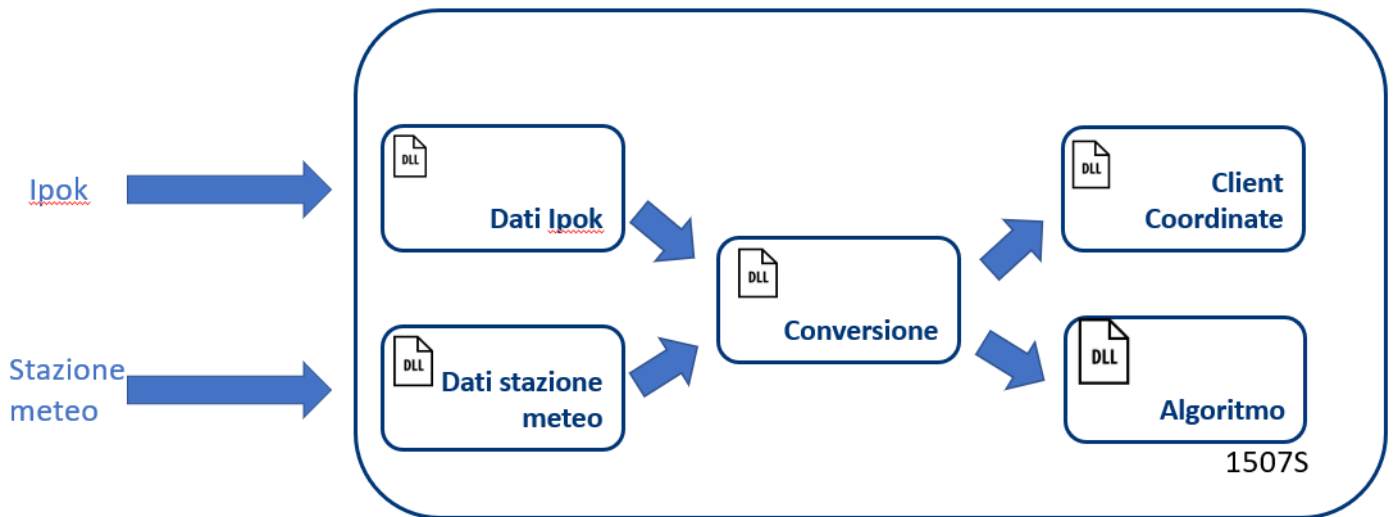


Figure 5.5: Internal structure of the Algorithm Module

Weather station data DLL: similarly to the previous block, the weather station sends data relating to the wind direction to the Siemens PC via TCP/IP.

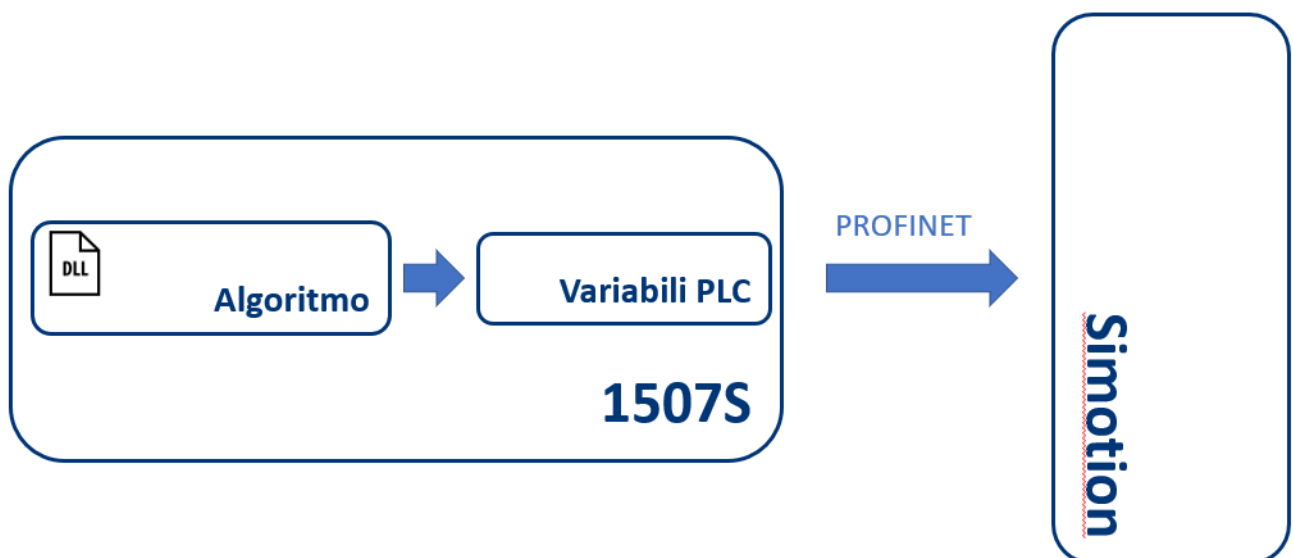


Figure 5.6: Profinet Interconnection

Conversion DLL: this FB receives the inputs from the Ipok Data and Weather Station Data DLLs to switch from global coordinates to local coordinates. These coordinates will then be sent to the Algorithm DLL and Coordinate Client DLL.

DLL Algorithm: generates the length differential of the ropes and the reference torque to send to the Simotion, which will pilot the sail. The outputs of the Algorithm DLL are saved in a PLC variable and sent via PROFINET to Simotion, thus working in deterministic and cyclical times.

DLL CLIENT COORDINATE: receives the data relating to the position of the Kite and sends it via TCP/IP to Unity on Windows to create a graphic.

The Coordinate Client is a TCP/IP socket client implemented in C++.

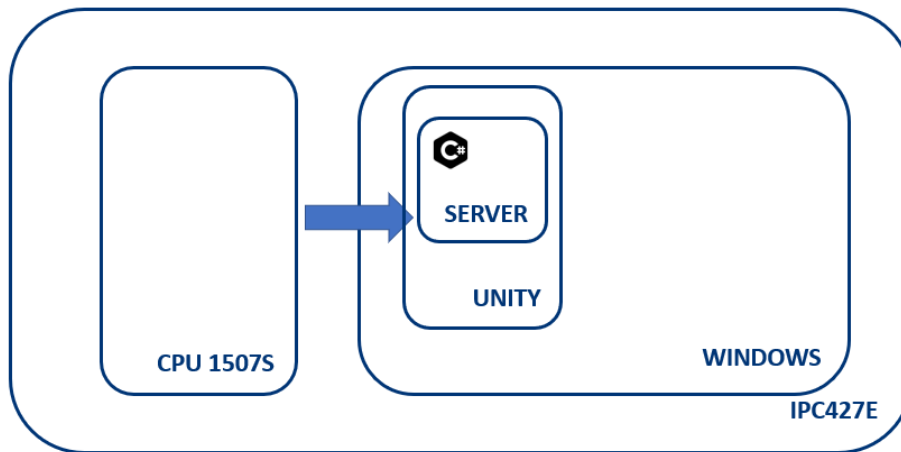


Figure 5.7: From Algorithm module to the Graphics module

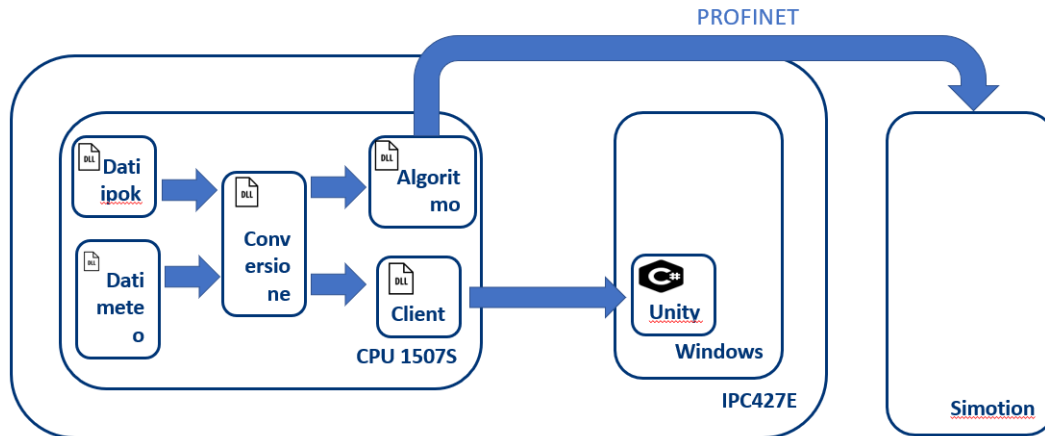


Figure 5.8: Overall functional blocks

5.3.2 Graphics

The graphics will be developed in Unity in the Windows operating system(as described in chapter 4). It was decided to use this program as it is a high level program, it has various features making it widely used in the world of graphics.

In this environment, a script will be implemented in Csharp, which will act as a TCP/IP Server which will receive the data via TCP from the CPU and will be fed to Unity, which will recreate the kite trajectory in 3D.

CHAPTER 6

Simulation and future steps

6.1 Simulation

In this chapter, the results obtained from the Matlab simulation will be compared with the model created in C++. This step is crucial, as for the model to be reliable enough to autonomously control the sail, the difference between the two models must be minimal.

To compare the results of the two methods, a program was created that generates outputs starting from a text file (using the same algorithm). These outputs will then be compared with the data from Matlab.

Specifically, the file consists of four rows, with each row containing data relevant for data processing:

-The first row contains θ , ϕ , ThetaDer ,PhiDer

-The second row contains the lengths and velocities of the cables.

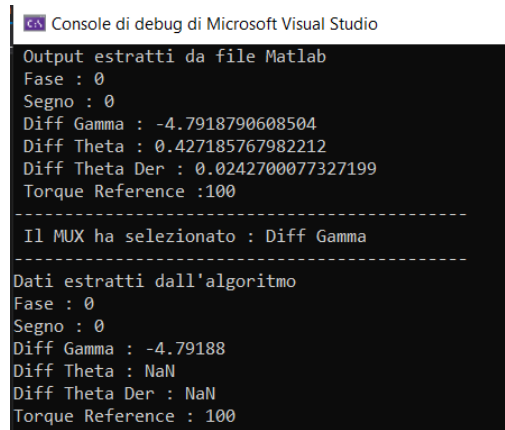
The third row contains the values of θ , ϕ from the previous iteration (this data is required because the program operates in an "open-loop" manner concerning the PID section)

-The last row contains the outputs generated by Matlab, which will then be compared with the results from the algorithm.

```
0.572814232017788;-0.258937241170999;-0.0242700077327199;0.014718130338132
-767.492762605433;0.00939312910642622;-761.427728693183;16.9911192827154
0;0;3.43364212352032;2.6645443452308;-2.12363168066743;-4.78817602589823;1;0;0.426943568393237;0.0242029220334166;1;100
0;0;3.43677678892894;2.67026802544889;-2.12161103540151;-4.7918790608504;1;0;0.427185767982212;0.0242700077327199;1;100
```

Figure 6.1: Input file example

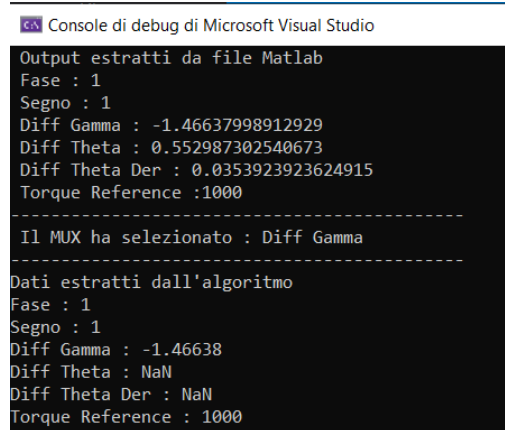
The program will "print" the results generated by the algorithm and compare them with the Matlab model. The goal is to obtain results with a precision up to the first decimal place.



```
Console di debug di Microsoft Visual Studio
Output estratti da file Matlab
Fase : 0
Segno : 0
Diff Gamma : -4.7918790608504
Diff Theta : 0.427185767982212
Diff Theta Der : 0.0242700077327199
Torque Reference :100
-----
Il MUX ha selezionato : Diff Gamma
-----
Dati estratti dall'algoritmo
Fase : 0
Segno : 0
Diff Gamma : -4.79188
Diff Theta : NaN
Diff Theta Der : NaN
Torque Reference : 100
```

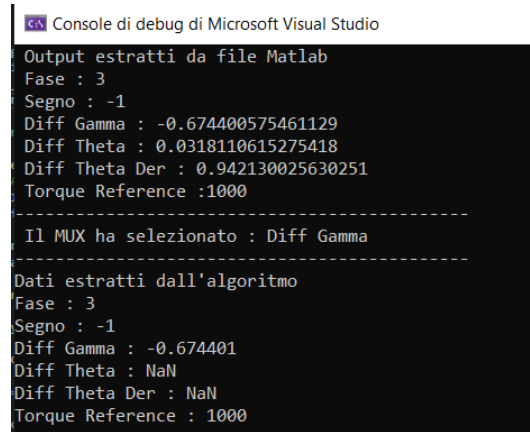
Figure 6.2: Output file example

To verify whether the algorithm is processing correctly, various moments were considered (including critical moments). In particular, the moments when the sail changes its "target" or transitions between phases were taken into account. This approach allows for the evaluation of the model's reliability. To achieve this, several input files were created to assess the performance of the algorithm throughout all phases of the flight.



```
Console di debug di Microsoft Visual Studio
Output estratti da file Matlab
Fase : 1
Segno : 1
Diff Gamma : -1.46637998912929
Diff Theta : 0.552987302540673
Diff Theta Der : 0.0353923923624915
Torque Reference :1000
-----
Il MUX ha selezionato : Diff Gamma
-----
Dati estratti dall'algoritmo
Fase : 1
Segno : 1
Diff Gamma : -1.46638
Diff Theta : NaN
Diff Theta Der : NaN
Torque Reference : 1000
```

Figure 6.3: Changing from phase 0 to 1



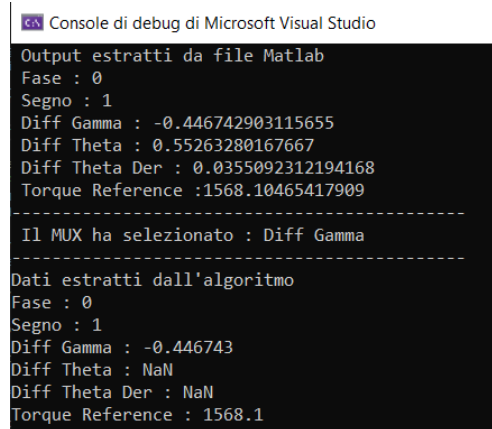
```

Console di debug di Microsoft Visual Studio

Output estratti da file Matlab
Fase : 3
Segno : -1
Diff Gamma : -0.674400575461129
Diff Theta : 0.0318110615275418
Diff Theta Der : 0.942130025630251
Torque Reference : 1000
-----
Il MUX ha selezionato : Diff Gamma
-----
Dati estratti dall'algoritmo
Fase : 3
Segno : -1
Diff Gamma : -0.674401
Diff Theta : NaN
Diff Theta Der : NaN
Torque Reference : 1000

```

Figure 6.4: Changing from phase 2 to 3 with a negative sign



```

Console di debug di Microsoft Visual Studio

Output estratti da file Matlab
Fase : 0
Segno : 1
Diff Gamma : -0.446742903115655
Diff Theta : 0.55263280167667
Diff Theta Der : 0.0355092312194168
Torque Reference : 1568.10465417909
-----
Il MUX ha selezionato : Diff Gamma
-----
Dati estratti dall'algoritmo
Fase : 0
Segno : 1
Diff Gamma : -0.446743
Diff Theta : NaN
Diff Theta Der : NaN
Torque Reference : 1568.1

```

Figure 6.5: Torque reference value

Unlike MATLAB, the algorithm does not always generate all outputs, which explains why some values appear as "NaN". As described in the algorithm chapter, each flight phase produces only specific outputs, meaning that non-relevant values are not considered. Instead, based on the kite's phase, it determines which values to calculate. In fact, only the relevant data for that particular moment are transmitted to SiMotion. Logically, this is as if a multiplexer were present in the algorithm, deciding which values to consider based on the inputs.

As can be observed from the various outputs, the algorithm developed accurately reflects the data processed by MATLAB, ensuring that the results align with the expected values and demonstrating the consistency between the two systems. This correlation further validates the algorithm's ability

to replicate the same data processing and output generation, similar to the MATLAB implementation.

6.2 Future steps

The next steps will involve configuring the Siemens machine (so far, work has been done on a general-purpose Windows machine). After that, field tests will be conducted, connecting the entire system to the sail and comparing in real-time the data generated by the algorithm with the position of the sail as controlled by a pilot. This will allow for the evaluation of the algorithm's performance (based on the real data transmitted by the sail) and for examining the discrepancies between the automatic pilot model and the manual pilot model during critical moments (such as target changes, phase changes, etc.). Additionally, the graphics created in Unity will be highly useful, as the real-time display will allow for visual verification of the sail's position in Unity and its actual position in the "sky" with the naked eye.

6.3 Collaborations and Acknowledgments

I would like to express my heartfelt thanks to the Motion Engineering team and Kyte Energy for the opportunity they provided in bringing this project to fruition.

CHAPTER 7

Appendix

7.1 ENU reference frame

ENU stands for East-North-Up and is a local coordinate system used to describe positions relative to a specific reference point, typically on the Earth's surface.

In the ENU system: The East axis points towards the east from the reference point (positive towards the east), The North axis points towards the north from the reference point (positive towards the north), The Up axis points upwards, perpendicular to the Earth's surface (positive upwards, away from the Earth's center).

The ENU system is commonly used in geodesy, navigation, and in applications where local positioning relative to a specific point is needed. It is often used to simplify calculations and analyses involving geographic locations, particularly in applications like GPS or robotics.

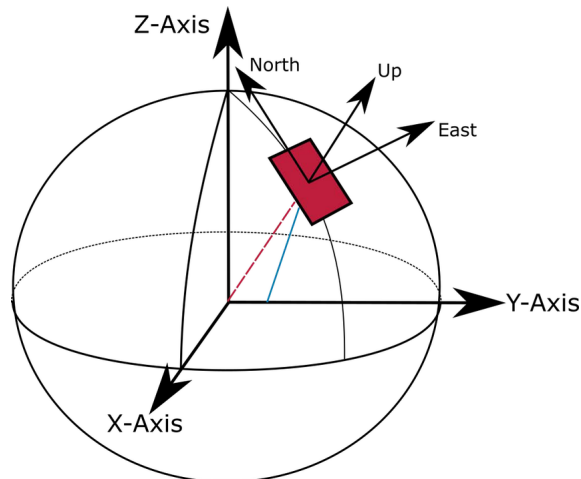


Figure 7.1: ENU reference frame example

7.2 UDP and TCP

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) both are protocols of the Transport Layer Protocols. TCP is a connection-oriented protocol whereas UDP is a part of the Internet Protocol suite, referred to as the UDP/IP suite. Unlike TCP, it is an unreliable and connectionless protocol. In this article, we will discuss the differences between TCP and UDP.

What is Transmission Control Protocol (TCP)?

TCP (Transmission Control Protocol) is one of the main protocols of the Internet protocol suite. It lies between the Application and Network Layers which are used in providing reliable delivery services. It is a connection-oriented protocol for communications that helps in the exchange of messages between different devices over a network. The Internet Protocol (IP), which establishes the technique for sending data packets between computers, works with TCP.

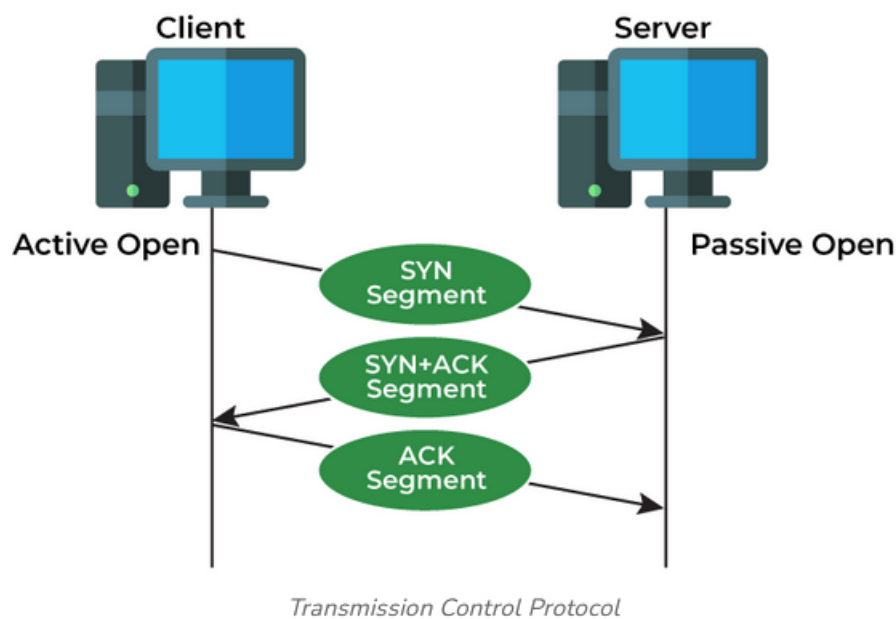


Figure 7.2: TCP Schema

Features of TCP

TCP keeps track of the segments being transmitted or received by assigning numbers to every single one of them.

Flow control limits the rate at which a sender transfers data. This is done to ensure reliable delivery.

TCP implements an error control mechanism for reliable data transfer. TCP takes into account the level of congestion in the network.

Applications of TCP

World Wide Web (WWW) : When you browse websites, TCP ensures reliable data transfer between your browser and web servers.

Email : TCP is used for sending and receiving emails. Protocols like SMTP (Simple Mail Transfer Protocol) handle email delivery across servers. File Transfer Protocol (FTP) : FTP relies on TCP to transfer large files securely. Whether you're uploading or downloading files, TCP ensures data integrity.

Secure Shell (SSH) : SSH sessions, commonly used for remote administration, rely on TCP for encrypted communication between client and server. Streaming Media : Services like Netflix, YouTube, and Spotify use TCP to stream videos and music. It ensures smooth playback by managing data segments and retransmissions.

Advantages of TCP

It is reliable for maintaining a connection between Sender and Receiver. It is responsible for sending data in a particular sequence. Its operations are not dependent on Operating System . It allows and supports many routing protocols. It can reduce the speed of data based on the speed of the receiver.

Disadvantages of TCP

It is slower than UDP and it takes more bandwidth. Slower upon starting of transfer of a file. Not suitable for LAN and PAN Networks. It does not have a multicast or broadcast category. It does not load the whole page if a single data of the page is missing.

What is User Datagram Protocol (UDP)?

User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as the UDP/IP suite. Unlike TCP, it is an unreliable and connectionless protocol. So, there is no need to establish a connection before data transfer. The UDP helps to establish low-latency and loss-tolerating connections establish over the network. The UDP enables process-to-process communication.

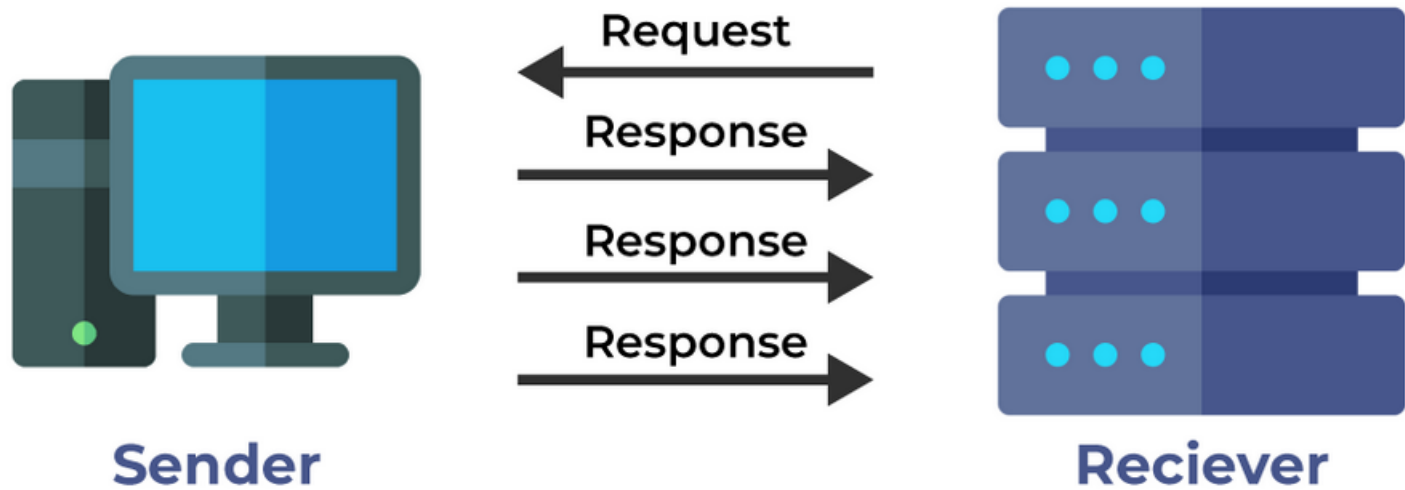


Figure 7.3: UDP Schema

Features of UDP

Used for simple request-response communication when the size of data is less and hence there is lesser concern about flow and error control. It is a suitable protocol for multicasting as UDP supports packet switching. UDP is used for some routing update protocols like RIP(Routing Information Protocol). Normally used for real-time applications which can not tolerate uneven delays between sections of a received message.

Application of UDP

Real-Time Multimedia Streaming : UDP is ideal for streaming audio and video content. Its low-latency nature ensures smooth playback, even if occasional data loss occurs.

Online Gaming : Many online games rely on UDP for fast communication between players.

DNS (Domain Name System) Queries : When your device looks up domain names (like converting “www.example.com” to an IP address), UDP handles these requests efficiently .

Network Monitoring : Tools that monitor network performance often use UDP for lightweight, rapid data exchange.

Multicasting : UDP supports packet switching, making it suitable for multicasting scenarios where data needs to be sent to multiple recipients simultaneously.

Routing Update Protocols : Some routing protocols, like RIP (Routing Information Protocol), utilize UDP for exchanging routing information among routers.

Advantages of UDP

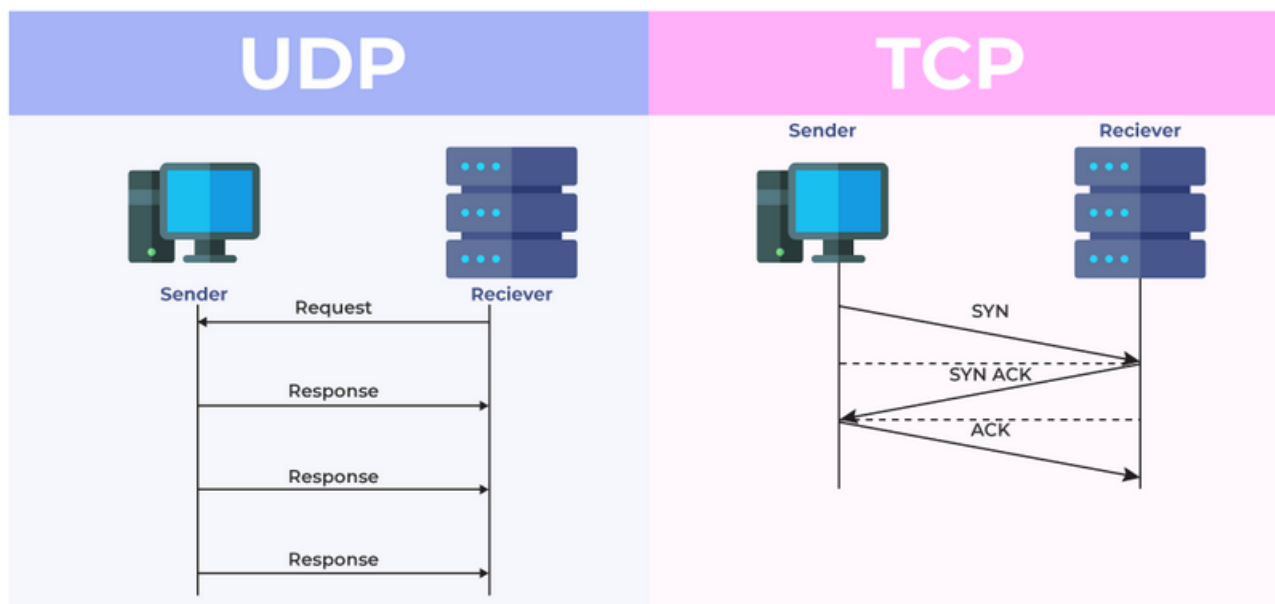
It does not require any connection for sending or receiving data. Broadcast and Multicast are available in UDP. UDP can operate on a large range of networks. UDP has live and real-time data. UDP can deliver data if all the components of the data are not complete.

Advantages of UDP

We can not have any way to acknowledge the successful transfer of data. UDP cannot have the mechanism to track the sequence of data. UDP is connectionless, and due to this, it is unreliable to transfer data. In case of a Collision, UDP packets are dropped by Routers in comparison to TCP. UDP can drop packets in case of detection of errors.

Which Protocol is Better: TCP or UDP?

The answer to this question is complex, as it depends entirely on the type of work being done and the nature of the data being transmitted. UDP is preferable for real-time applications like online gaming, as it minimizes lag. On the other hand, TCP is ideal for transferring data such as photos or videos, as it guarantees the accuracy and integrity of the data being sent. In general, both TCP and UDP are useful in the context of the work assigned by us. Both have advantages upon the works we are performing, that's why it is difficult to say, which one is better.



Difference Between TCP and UDP

Figure 7.4: UDP vs TCP

7.3 DLL

A DLL is a library that contains code and data that can be used by more than one program at the same time. For example, in Windows operating systems, the Comdlg32 DLL performs common dialog box related functions. Each program can use the functionality that is contained in this DLL to implement an Open dialog box. It helps promote code reuse and efficient memory usage.

By using a DLL, a program can be modularized into separate components. For example, an accounting program may be sold by module. Each module can be loaded into the main program at run time if that module is installed. Because the modules are separate, the load time of the program is faster. And a module is only loaded when that functionality is requested.

Additionally, updates are easier to apply to each module without affecting other parts of the program. For example, you may have a payroll program, and the tax rates change each year. When these changes are isolated to a DLL, you can apply an update without needing to build or install the whole program again.

The following list describes some of the advantages that are provided when a program uses a DLL:

Uses fewer resources

When multiple programs use the same library of functions, a DLL can reduce the duplication of code that is loaded on the disk and in physical memory. It can greatly influence the performance of not just the program that is running in the foreground, but also other programs that are running on the Windows operating system.

Promotes modular architecture

A DLL helps promote developing modular programs. It helps you develop large programs that require multiple language versions or a program that requires modular architecture. An example of a modular program is an accounting program that has many modules that can be dynamically loaded at run time.

Eases deployment and installation

When a function within a DLL needs an update or a fix, the deployment and installation of the DLL does not require the program to be relinked with the DLL. Additionally, if multiple programs use the same DLL, the multiple programs will all benefit from the update or the fix. This issue may more frequently occur when you use a third-party DLL that is regularly updated or fixed.

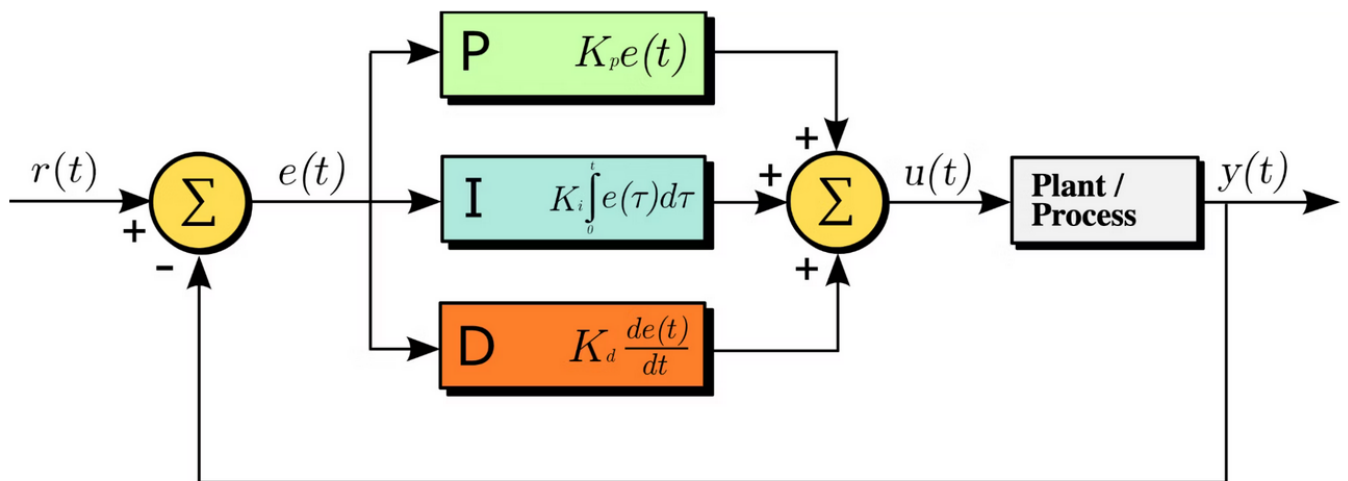
7.4 PID

PID controllers represent a sophisticated feedback mechanism vital for controlling dynamic systems. At their core, they operate using three basic terms: Proportional (P), Integral (I), and Derivative (D). Each term uniquely modulates the output signal based on the difference between the desired setpoint and the actual measured value, commonly known as the error.

The Proportional (P) term responds to the present error, generating an output proportional to its magnitude. By applying immediate corrective action, the P term minimizes errors quickly.

The Integral (I) term addresses any persistent errors or long-term deviations from the setpoint by accumulating the error over time. By integrating the error signal, the I term ensures that the system approaches and maintains the setpoint accurately, eliminating steady-state errors.

The Derivative (D) term anticipates future changes in the error by evaluating its rate of change. This approach dampens oscillations and stabilizes the system, especially during transient responses.



PID Block Diagram, Arturo Urquizo, CC BY-SA 3.0 via Wikimedia Commons

Figure 7.5: Generic PID Schema

A PID controller continuously calculates an error signal as the difference between a desired setpoint (the target value) and the current process variable (the measured value). Based on this error signal, the controller adjusts the system's control input to minimize the error and maintain the process vari-

able close to the setpoint.

Here's how it works in more detail:

Error Calculation: The PID controller continuously calculates the error signal as the difference between the desired set point and the current process variable.

Proportional Control: The proportional (P) term responds to the current error by producing an output proportional to the magnitude of the error.

Integral Control: The integral (I) term responds to the cumulative sum of past errors and aims to eliminate any steady-state error. It is calculated as the integral of the error over time.

Derivative Control: The derivative (D) term responds to the rate of change of the error and helps dampen rapid changes in the system. It is calculated as the derivative of the error over time.

Control Output: The control output is the sum of the proportional, integral, and derivative terms.

Adjustment of Control Input: The control output is applied as the input to the system being controlled. It adjusts system parameters such as valve positions, motor speeds, or heating elements to bring the process variable closer to the set point.

Feedback Loop: The process variable is continuously measured and fed back to the controller, closing the control loop. The controller adjusts the control input based on the feedback, aiming to minimize the error and maintain the process variable at the setpoint.

7.5 Simotion

Simotion is a motion control system developed by Siemens. It is part of the Siemens Totally Integrated Automation (TIA) portfolio and is designed to provide high-performance motion control solutions for industrial automation applications. Simotion is used for controlling and optimizing the movement of machines and processes in various industries, such as manufacturing, robotics, packaging, and more.

Simotion is essentially a motion control system that integrates various motion control tasks, such as positioning, velocity control, and torque control. It allows for precise and flexible control of drive systems, enabling machinery to operate efficiently and accurately. It can handle both simple and complex motion profiles, offering robust control over different types of motion systems, including both synchronous and asynchronous motors.

Simotion is primarily used to: Control and optimize motion in machinery: It is employed to control the movements of motors, actuators, and other devices in an automated environment.

Enhance production processes: By providing precise motion control, it can improve the accuracy, speed, and overall performance of industrial machines. **Integrate various automation components:** It

can integrate with other Siemens automation systems (like PLCs, HMI, and SCADA) to create a comprehensive automation solution.

Simplify programming and maintenance: Simotion provides user-friendly tools to design, configure, and manage motion control applications, which simplifies the programming and maintenance tasks.

Simotion is structured into several components that work together to provide motion control functionality:

Simotion Controllers: These are the central units of the system, responsible for controlling the motion processes. Simotion controllers can be programmable or configured using software tools.

Simotion Software: This includes the Simotion Scout programming and configuration tool, which is used for setting up and programming the motion control system. It allows the user to define motion profiles, set parameters, and visualize system behavior.

Motion Control Modules: These are used to interface with the motors and drives, providing the necessary hardware for controlling different motion parameters (e.g., speed, position, torque).

Simotion Drives: These include the drive systems that control the motor's operation based on commands from the Simotion controller. They allow for precise control of motor functions like speed, torque, and position.

Communication Modules: Simotion uses communication networks (such as Profinet, Ethernet, or Profibus) to connect various automation components and enable data exchange between the controller, drives, and other devices in the system.

HMI (Human-Machine Interface): For monitoring and interacting with the motion control system, Simotion can be integrated with HMIs, allowing operators to visualize machine status, adjust parameters, and control motion sequences.

Key Features of Simotion are :

High-precision motion control: It provides advanced control over positioning, speed, and torque, making it ideal for applications that require high levels of precision. **Flexible programming:** It supports both programming in IEC 61131-3 languages (like Ladder Logic, Structured Text) and motion-specific function blocks for easy integration into control systems.

Scalability: Simotion can be scaled to meet the needs of small machines or large, complex systems with multiple axes of motion.

Multi-axis control: It can handle systems with multiple axes (motors), such as synchronized movement in robotics and packaging systems.

Real-time performance: Simotion ensures high-speed control and real-time response to changes in the system, critical for high-performance applications. **Integration with other Siemens automation products:** It integrates seamlessly with other Siemens products, allowing for a fully integrated automation

solution.

In other words, Simotion is a versatile and powerful motion control system designed to provide precise control over industrial machinery and processes. Its flexibility, scalability, and high-performance features make it suitable for a wide range of automation applications. Through the integration of controllers, drives, and communication systems, Simotion enables industries to optimize their production processes and improve machine performance.

7.6 PROFINET

PROFINET is an open Industrial Ethernet solution based on international standards. It is a communication protocol designed to exchange data between controllers and devices in an automation setting. It was introduced in the early 2000s and is the most well-adopted Industrial Ethernet solution.

Since PROFINET is an open standard, hundreds of manufacturers have developed PROFINET products, such as PLCs, PACs, Drives, Robots, Proxies, IOs, and diagnostic tools.

PROFINET defines cyclic and acyclic communication between components, including diagnostics, functional safety, alarms, and additional information. To link all of those components, PROFINET employs standard Ethernet for its communication medium. Ethernet cables connect PROFINET components within a network, allowing other Ethernet protocols to coexist within the same infrastructure. Besides PROFINET, you can employ other Ethernet-based protocols to complement the network, such as OPC UA, SNMP, MQTT, or HTTP.

Industrial automation environments often require high-speeds and deterministic communication. Deterministic communication means delivering messages exactly when they are expected.

PROFINET must ensure messages are delivered with the appropriate speed and determinism depending on the task. Not all applications require the same performance. For example, loading configuration data for a process instrument can take several minutes without affecting production. On the other hand, a communication delay of just a few milliseconds between a PLC and a high-speed VFD can significantly impact the process.

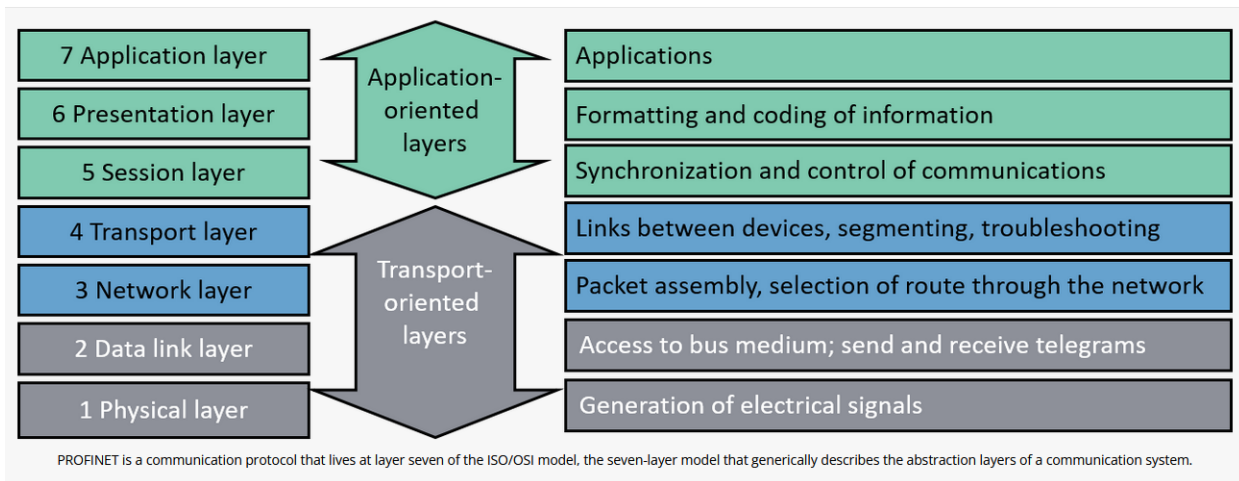


Figure 7.6: Profinet Schema

To ensure appropriate performance, PROFINET delivers data through the following communication channels:

TCP/IP (or UDP/IP)

PROFINET Real-Time (RT)

PROFINET Isochronous Real-Time (IRT)

Time Sensitive Networking (TSN)

PROFINET can employ TCP/IP or UDP/IP communication for non-time-critical tasks, such as configuration and parameterization. Due to the added latency and jitter associated with IP-based communication, this method is unsuitable for time-critical tasks.

PROFINET employs a real-time channel to deliver data in a fast and deterministic fashion for time-critical applications.

This is how PROFINET real-time works: Standard Ethernet frames have a field called an EtherType, which indicates the type of protocol used. PROFINET real-time communication is set to have an EtherType of 0x8892. Upon arrival at the destination node, the frame is directed to the PROFINET application immediately. The data goes directly from Ethernet, Layer 2, to PROFINET, Layer 7. It skips the TCP/IP layers and avoids the variable time it could take to be processed as such. Thus, communication speed and determinism improve significantly. PROFINET real-time fulfills the requirements of the vast majority of time-critical applications. The overall performance will depend on your network's design, but generally, you can achieve cycle times of 250 μ s to 512 ms.

7.7 Socket Programming

Socket programming is a way of enabling communication between computers or devices over a network using a standardized interface. It provides a mechanism for creating networked applications that can exchange data between a client and a server. Socket programming is typically used in various network protocols (like TCP/IP) and is crucial in enabling features like web browsing, file transfers, and real-time communication.

Here's a theory-oriented explanation of Socket Programming:

1. What is a Socket?

A socket is an endpoint for communication between two machines over a network. It serves as an interface for sending and receiving data. In simpler terms, a socket is a combination of an IP address and a port number. It helps applications communicate using the TCP (Transmission Control Protocol) or UDP (User Datagram Protocol), which are the two main types of protocols for network communication.

TCP (Transmission Control Protocol): A connection-oriented protocol that ensures reliable data transmission. It establishes a reliable, stream-based connection between the client and server. It guarantees that packets of data arrive in the correct order and retransmits any lost packets.

UDP (User Datagram Protocol): A connectionless protocol that is faster but does not guarantee reliable delivery. It is used when speed is more important than reliability, like in streaming or real-time applications.

2. Types of Sockets

Stream Sockets (TCP sockets): These sockets are used for communication using the TCP protocol. They ensure reliable, two-way, byte stream communication between the client and server.

Datagram Sockets (UDP sockets): These sockets use UDP, a simpler, connectionless protocol that does not ensure reliable transmission. They send packets (datagrams) between the client and server without establishing a dedicated connection.

3. Basic Socket Programming Steps

Here's an overview of the steps involved in socket programming. This includes both client-side and server-side operations.

Server-side Steps:

Create a Socket: The server creates a socket to listen for incoming connections.

Example: `serversocket = socket.socket(socket.AF_INET, socket.SOCKSTREAM)` **AF_INET:** Specifies the IPv4 protocol.

SOCKSTREAM: Specifies the TCP protocol.

Bind the Socket: Bind the socket to a specific IP address and port number. This step links the socket to the local machine and makes it ready to accept incoming connections. Example: `serversocket.bind(('localhost', 12345))`

Listen for Connections: The server listens for incoming client connections. Example: `serversocket.listen(5)` (5 is the maximum number of pending connections in the queue).

Accept Connections: When a client attempts to connect, the server accepts the connection and establishes a new socket dedicated to communication with that client. Example: `clientsocket, clientaddress = serversocket.accept()`

Data Exchange: The server and client can now exchange data over the socket. Example: `clientsocket.send(data)` and `data = clientsocket.recv(1024)`.

Close the Socket: After communication is done, the server closes the socket. Example: `clientsocket.close()` and `serversocket.close()`.

Client-side Steps:

Create a Socket: The client creates a socket that is used to connect to the server. Example: `clientsocket = socket.socket(socket.AF_INET, socket.SOCKSTREAM)`

Connect to the Server: The client connects to the server by specifying the server's IP address and port number. Example: `clientsocket.connect(('localhost', 12345))`.

Data Exchange: The client can send and receive data from the server. Example: `clientsocket.send(data)` and `data = clientsocket.recv(1024)`.

Close the Socket: After communication is done, the client closes the socket. Example: `clientsocket.close()`

4. Socket Programming Functions

Common functions and methods used in socket programming include:

`socket()`: Creates a new socket. `bind()`: Binds the socket to a particular address and port. `listen()`: Makes the socket listen for incoming connections (used by the server). `accept()`: Accepts a connection from a client (used by the server). `connect()`: Connects to a server (used by the client). `send()`: Sends data through the socket. `recv()`: Receives data from the socket. `close()`: Closes the socket.

5. Socket Addressing

Sockets are typically identified by:

IP address: A unique identifier for a machine on the network. **Port number:** A number used to identify a specific process or service on the machine. For example, a socket may be identified by ('localhost', 8080), where localhost is the IP address and 8080 is the port number.

6. Blocking vs Non-blocking Mode

Blocking Mode: In this mode, the socket operations like `recv()` or `accept()` block the program's ex-

ecution until the operation is complete. This means the program waits until data is received or a connection is accepted. Non-blocking Mode: In this mode, the socket operations do not block the program. If there is no data available, or the connection is not ready, the operation will return immediately, allowing the program to continue running.

7. Error Handling

Socket programming involves handling errors that may occur during network communication, such as: Connection errors (unable to connect to the server). Timeout errors (data transfer takes too long). Resource errors (insufficient resources to create a socket). Error handling is crucial to ensuring the reliability of a network application.

CHAPTER 8

References

- <https://techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/>
- https://www.repubblica.it/green-and-blue/2022/05/26/news/energia_eolica_ite_v_ele-351196827/
- <https://www.c-sharpcorner.com/article/socket-programming-in-C-Sharp/>
- <https://vichargrave.github.io/programming/tcp-ip-network-programming-design-patterns-in-cpp/>
- <https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/>
- https://fagiano.faculty.polimi.it/docs/papers/HAWP_1EEETEC2010.pdf
- <https://fagiano.faculty.polimi.it/docs/papers/2022-AWEAR.pdf>
- <https://www.isa.org/intech-home/2022/february-2022/departments/what-you-may-not-know-about-softplcs>
- <https://us.profinet.com/profinet-explained/>
- <https://www.geeksforgeeks.org/differences-between-tcp-and-udp/>
- <https://dewesoft.com/blog/what-is-pid-controller>
- <https://stackoverflow.com/questions/>
- <https://www.geeksforgeeks.org/socket-programming-in-cpp/>
- <https://www.vectornav.com/resources/inertial-navigation-primer/math-fundamentals/math-refframes>
- <https://kitenrg.com/>
- <https://support.industry.siemens.com/cs/document/109479653/simotion-documentation?dti=0lc=en-IT>
- <https://support.industry.siemens.com/cs/document/109751885/catalog-pm-21-simotion-equipment-for-production-machines?dti=0lc=en-HN>
- https://cache.industry.siemens.com/dl/files/629/109744629/att908763/v1/SIMOTION_documentation_o_verview_en-

US.pdf

–https : //cache.industry.siemens.com/dl/files/081/27035081/att_11478/v1/C2xxOperating_en –

US.pdf

–https : //x.company/projects/makani/

–https : //en.wikipedia.org/wiki/Kite_power

–https : //www.energy.gov/eere/wind/advantages – and – challenges – wind – energy

–https : //www.sciencedirect.com/journal/renewable – energy

–https : //www.siemens.com/global/en/home.html

–https : //en.wikipedia.org/wiki/East–north–up_coordinates

–https : //docs.python.org/3/library/socket.html

–https : //www.geeksforgeeks.org/socket – programming – python/

–https : //www.tutorialspoint.com/python/python_networking.htm

–https : //en.wikipedia.org/wiki/Socket_(computer_networking)

–https : //www.profibus.com/profinet/

–https : //en.wikipedia.org/wiki/PROFINET

–https : //new.siemens.com/global/en/products/automation/topic – areas/profinet.html

–https : //en.wikipedia.org/wiki/PID_controller

–https : //www.mathworks.com/help/control/ref/pid.html

–https : //en.wikipedia.org/wiki/Transmission_Control_Protocol

–https : //en.wikipedia.org/wiki/User_Datagram_Protocol

–https : //learn.microsoft.com/en – us/troubleshoot/windows – client/setup – upgrade – and – drivers/dynamic – link – library