Proyecto Fin de Carrera
Ingeniería Electrónica Robótica y Mecatrónica

Desarrollo de sistema de calibrado automático de espejos de captador solar tipo Fresnel

Autor: Javier Jesús Laó Amores
Tutor: Juan Manuel Escaño González
Cotutor: Adolfo J. Sánchez Del Pozo Fernández

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Carrera
Ingeniería Electrónica Robótica y Mecatrónica

Desarrollo de sistema de calibrado automático de espejos de captador solar tipo Fresnel

Autor:
Javier Jesús Laó Amores

Tutor:
Juan Manuel Escaño González
Cotutor:
Adolfo J. Sánchez Del Pozo Fernández

Dep. de Ingeniería de sistemas y automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: Desarrollo de sistema de calibrado automático de espejos de captador solar tipo Fresnel

Autor: Javier Jesús Laó Amores

Tutor: Juan Manuel Escaño González

Cotutor: Adolfo J. Sánchez Del Pozo Fernández

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Dedico este proyecto a mi familia, por servirme de apoyo y proporcionarme los medios necesarios para realizar estos estudios. Sin su apoyo, paciencia y comprensión no hubiera sido posible.

A la Escuela Técnica Superior de Ingenieros de Sevilla, por la oportunidad que me brindó a la hora de realizar una estancia en el extranjero, descubriendo que es posible encontrar una familia allá donde vayas.

A todos los profesores, y de una forma especial a Juan Manuel Escaño González y Adolfo J. Sánchez del Pozo por haberme brindado la oportunidad de realizar este proyecto.

A mis amigos, que me han apoyado y animado en todos los momentos vividos en la carrera, sobre todo en los malos, cuando siempre han sido un hombro donde apoyarse.

A mis compañeros de universidad, con los cuales ha sido posible salir de los baches y seguir adelante siempre con una sonrisa y una anécdota que contar. Gracias a ellos, el edificio de la escuela ha sido más un hogar que una universidad.

Mi agradecimiento a todos ellos,

Javier Jesús Laó Amores

Sevilla, 2020

Resumen

El planteamiento original de este proyecto fue volver a poner en funcionamiento la planta solar de tipo fresnel instalada en la azotea del edificio principal de la ETSI. Para ello se realizó un estudio previo para valorar el estado actual de la instalación.

Una vez realizado dicho estudio, se decidió actuar sobre los servomotores estropeados y la infraestructura de la comunicación entre el campo y el autómata programable. La contratación de estos trabajos fue interrumpida por la implantación del estado de alarma en todo el territorio nacional. Como consecuencia hubo que replantear el proyecto para poder continuarlo de forma no presencial y que los resultados obtenidos pudiesen contribuir a la posterior puesta en marcha de la planta.

Esta nueva orientación ha consistido en elaborar un simulador de la planta en un entorno que permitiese la comunicación ModBus sobre TCP/IP entre este y el software utilizado para la programación del autómata programable. Además, se ha mejorado el programa de calibración por uno automático que ha sido testeado en el propio simulador.

Abstract

In this project we will study the state of the linear fresnel reflector installed on the rooftop of the ETSI. Once the current status of the plant was checked, a new start up was necessary. For this purpose, a new automatic calibration program was developed. The calibration program makes use of a solar sensor to determine in which position the reflectors concentrate the maximum number of solar beams to the collector. However, it could not be tested directly in the solar plant due to the state of alarm initiated in March 2020 which did not allowed perform this task at the ETSI.

To overcome this issue, a simulator of the solar plant has been developed in the game engine Unity 3D. The main language used for the simulator was UnityScript and the configuration of the simulated linear fresnel reflector matches with the real one. The communication between the simulator and the calibration program in Unity Pro XL was executed through ModBusTCP/IP.

Finally, the calibration program was proved in the developed simulator achieving good results. The main sensor for the calibration was coded in three different ways and two different calibration programs were tested in order to decide which of them was the best combination. The result showed that the most appropriate way to calibrate the solar plant is to use the sensor based on incremental steps with the calibration program that detects the peak of the sensor output.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

ETSI	Escuela Técnica Superior de Ingeniería
PLC	Programmable Logic Controller
CAN	Controller Area Network
CiA	CAN in Automation
CANopen	Protocolo de comunicación de alto nivel basado en el bus CAN
TCP	Transmission Control Protocol
IP	Internet Protocol

1 INTRODUCCIÓN

Las energías renovables son cada día más importantes, ya que son energías limpias e inagotables que nos proporciona la naturaleza. Entre las muchas ventajas que tienen una de las más importantes es reducir el efecto invernadero que se produce por el uso de combustibles fósiles y así proteger nuestro planeta. Entre las energías renovables podemos encontrar la energía hidráulica, la energía eólica, la energía geotérmica, la energía mareomotriz, la biomasa o la energía solar.

1.1 Estado del arte

La energía solar es la más utilizada en Andalucía, ya sea en plantas solares térmicas o en plantas solares fotovoltaicas debido al clima de la zona que como se puede ver en Figura 1-1, es la comunidad autónoma con mayor número de horas solares.

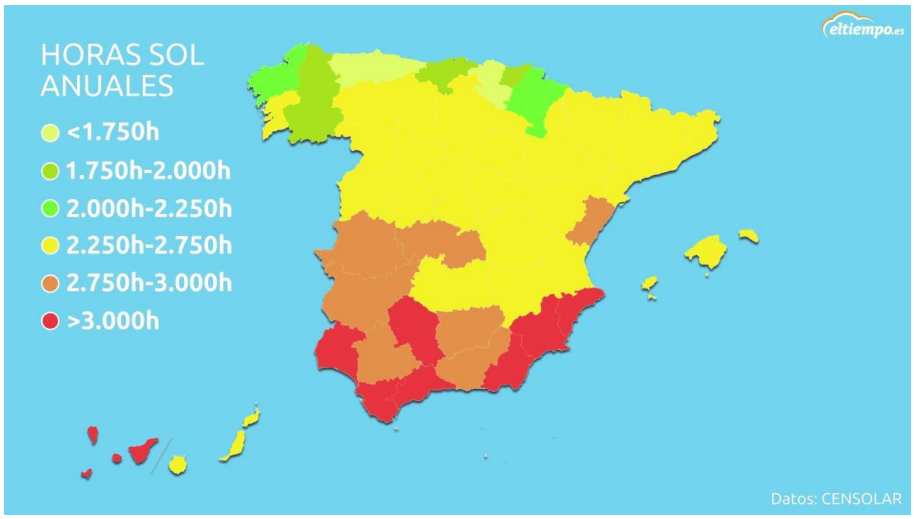

Figura 1-1.- Horas de sola anuales en España. [1]

Una forma de obtener la energía solar es mediante plantas solares. Como se ha mencionado anteriormente hay dos formas de obtención de esta energía mediante plantas solares fotovoltaicas que producen electricidad a partir de la radiación solar o mediante plantas solares térmicas que calientan un material, comúnmente agua que se podrá utilizar como agua caliente sanitaria o para la producción de electricidad. Dentro de las plantas solares térmicas podemos encontrar distintos modelos:

- Central térmica solar de torre: en la Figura 1-2 podemos ver que está compuesta por una torre central donde será concentrada la radiación solar a través de los subsistemas colectores situados sobre el terreno. Pueden alcanzar unas temperaturas mayores y por su modo de funcionamiento no es necesario construirlas sobre un terreno completamente plano, ya que los distintos subsistemas reflectores se pueden colocar a distintas alturas como puede ser una ladera y cada uno será orientado hacia la torre de forma independiente. La altura de la torre implica un mayor coste en la construcción y en el transporte de la energía ya que estará concentrada en lo alto de esta y tendrá que ser transportada hasta el inferior para poder ser distribuida.

Figura 1-2.- Central térmica solar de torre. [2]

- Planta solar de colectores cilindro parabólicos: como se observa en la Figura 1-3, el funcionamiento se basa en la forma parabólico de los colectores. La radiación es concentrada sobre el colector que transporta un fluido que será calentado a la salida del sistema será utilizado para obtener la energía que ha ido absorbiendo durante el recorrido. Pueden ser construidas a nivel del suelo, esto abarata mucho su precio, tienen una eficiencia bastante aceptable y permiten el uso de fluidos iónicos que permiten un mayor almacenaje de la energía. La forma de los espejos eleva el precio de estos.

Figura 1-3.- Central solar de colectores cilindro parabólicos. [3]

- Planta solar fresnel: este tipo de plantas son el mayor competidor de las plantas solares de colectores cilindro parabólicos, su funcionamiento es muy similar pero su construcción es más compacta y sus espejos al ser planas tiene un precio mucho mejor. En la Figura 1-4, se puede ver que requieren ser construidas sobre un terreno plano como las anteriores y que su mantenimiento es sencillo debido a la altura de la instalación.

Figura 1-4.- Planta solar fresnel. [4]

1.2 Justificación del proyecto

Obtener un elevado rendimiento conlleva mejorar la absorción de energía solar, esto tiene mucha importancia tanto en temas económicos como en medioambientales. En la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla se dispone de un captador solar tipo fresnel conectado a una máquina de absorción que entrega potencia a la climatización del edificio. Esta instalación se encontraba desconectada debido a los distintos fallos de sus componentes. Para una nueva puesta en marcha era necesario un buen sistema de calibración, el cual se desarrollará en este proyecto.

Como se ha comentado anteriormente para la nueva puesta a punto de la planta se buscaba arreglar los componentes electrónicos que fallaban y mejorar el código del autómata. En primer lugar, había que detectar que servomotores habían dejado de funcionar, una vez se realizó este paso, se comprobó que también existía un problema con el bus CANopen. El arreglo de ambas cosas se escapaba de los medios y conocimientos dispuestos para este proyecto, por lo que dependería de alguien externo. Sin embargo, la mejora del código del autómata si era posible, por lo que en el proyecto nos centraremos en el firmware de la planta.

1.3 Necesidad de un simulador

Debido a la situación acontecida a partir de marzo de 2020 con el estado de alarma, fue imposible continuar el proyecto en la planta solar de la ETSI, por ello se buscó una alternativa para poder continuar con el proyecto inicial. Se requería simular el comportamiento de una planta solar tipo fresnel, para poder establecer una comunicación con esta y poder continuar con el programa de la calibración. Además, este simulador podrá ser utilizado para implantar futuras mejoras en la planta solar de la ETSI. Para la simulación el software escogido fue Unity 3D debido a la alta versatilidad, sus licencias gratuitas y porque ya había sido utilizado anteriormente para establecer una comunicación ModBus sobre TCP/IP con otros softwares de Schneider Electric [5] [6].

2 CAPTADOR SOLAR FRESNEL

2.1 Captador solar fresnel

Un captador solar de tipo fresnel es una de las distintas plantas solares existentes en el mercado, el objetivo principal es reflejar los rayos solares sobre un largo receptor cilíndrico donde se generará calor. El receptor cilíndrico estará situado a lo largo de una línea recta, por esto, es denominado un sistema de foco lineal.

2.1.1 Funcionamiento captador solar tipo fresnel

El fin de un sistema de este tipo es la generación de energía, esta tiene lugar en forma de calor dentro del tubo cilíndrico por donde circulará un fluido que será el encargado de transportar ese calor a una máquina de absorción, donde se absorberá y se producirá frío. En lugar de una máquina de absorción se podría utilizar un intercambiador de calor donde en lugar de agua calentada llegaría vapor de agua que se utilizaría para producir electricidad a través de una turbina. La generación de este calor se produce por la reflexión de la radiación solar a través de una serie de superficies reflectoras orientadas hacia el tubo receptor. Para concentrar correctamente los rayos solares sobre la superficie receptora se utiliza un mecanismo de control como puede ser un autómata programable. El autómata será el encargado de conocer la trayectoria del sol y hacer funcionar una serie de motores para mantener el sistema de superficies reflectoras orientadas hacia el tubo receptor. Por tanto, el sistema se compondría de los siguientes elementos:

- Receptor: normalmente formado por uno o varios tubos situados en un plano superior y paralelo al de los dispositivos reflectores. Contiene el fluido que transporta el calor por la instalación. Se encuentra rodeado parcialmente por un reflector secundario.
- Concentradores solares: sistema generalmente formado por espejos ligeramente curvados encargado de concentrar la energía solar en el receptor.
- Motores: moverán los concentradores solares a partir de las señales recibas del sistema de control.
- Sistema de control: este dispositivo controlará cada concentrador de forma independiente para orientarlos hacia el receptor.
- Estructura: sujetará los diferentes planos de instalación.

Todos estos elementos se pueden observar en la Figura 2-1.

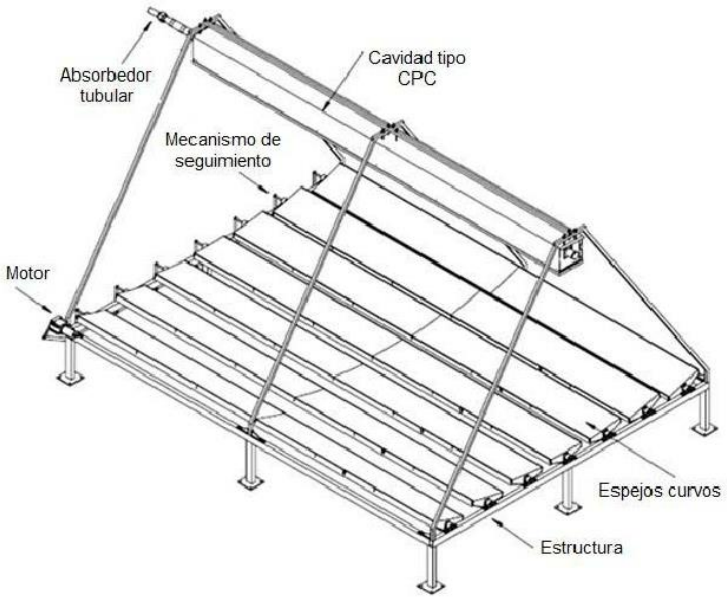

Figura 2-1.- Elementos planta solar tipo fresnel.

2.1.2 Ventajas del captador solar tipo fresnel

El competidor directo de este tipo de plantas solares son los captadores de tipo cilindro-parabólicos, a continuación, explicaremos cuales son las ventajas frente a estos [7]:

- En el caso de la fresnel el receptor es fijo, esto implica una mayor flexibilidad en la selección del fluido de transferencia térmica ya que no requiere de uniones móviles de alta presión.
- Mientras que el concentrador solar de un captador cilindro-parabólico es un único espejo con forma circular, en las plantas de tipo fresnel, hay múltiples espejos que solo son ligeramente curvos. Esto abarata el precio del montaje de la instalación y del arreglo en caso de rotura de algún espejo.
- Mayor aprovechamiento del terreno y minimización de la sobra producida por los reflectores sobre sus adyacentes.
- Fácil acceso a los reflectores y partes móviles debido a su construcción cercana al suelo. Además, reduce las cargas ejercidas por el viento.
- Menores pérdidas térmicas y una mayor temperatura de funcionamiento.
- Entre los fluidos se encuentran las sales fundidas que permiten el almacenamiento de la energía.

2.1.3 Instalación ETSI

La planta solar de tipo fresnel se encuentra instalada en la azotea de la Escuela Técnica Superior de Ingenieros de la Universidad de Sevilla (ETSI). La instalación se sitúa a 6° de longitud hacia el oeste y a 37,41° de latitud. La planta solar está situada a 12°3'1" al suroeste de la fachada sur y paralelamente a ésta [8].

Figura 2-2.- Localización planta solar en la ETSI.

El esquema general de la planta se encuentra en la Figura 2-3, en el podemos ver que la salida del sistema solar pasa por un tanque de almacenamiento y después llega a una máquina de absorción de doble efecto. El objetivo de la instalación es refrigerar el edificio a partir del frío producido en la máquina de absorción por medio del agua calentada por la energía solar. En caso de tener exceso de energía esta se almacena en el tanque mientras que si la energía producida no es suficiente se utilizará el quemador de Gas Natural instalado para generar la energía restante.

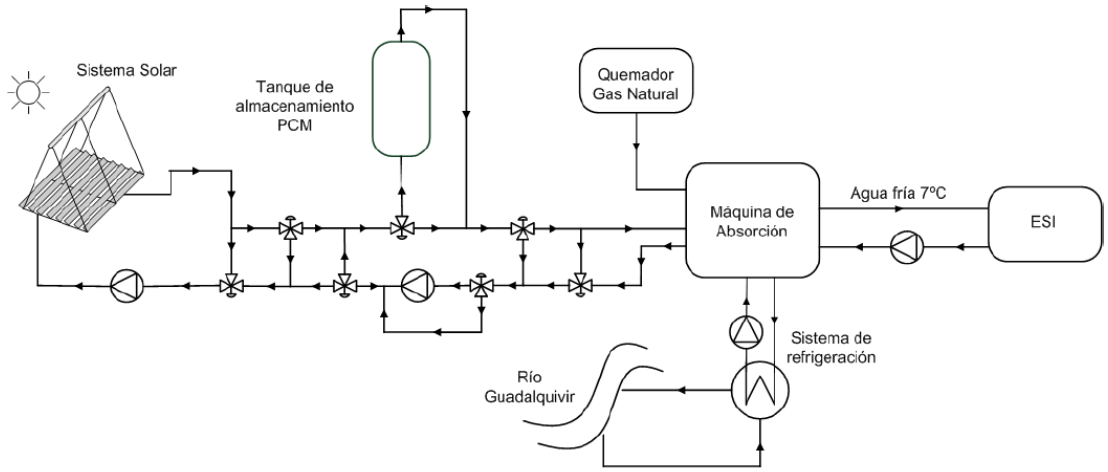

Figura 2-3.- Esquema general de la instalación.

Centrándonos en el sistema solar, las características de la planta se encuentran en la siguiente tabla:

Tabla 1.- Características del sistema de captación solar instalado en la ETSI [8].

Extensión de terreno ocupada	480 m^2
Superficie reflectora primaria total	352 m^2
Orientación de la planta	Este-Oeste
Número de líneas receptoras	1
Longitud de la línea receptora	64 m
Tipo de receptor	De cavidad con reflector secundario y cubierta de vidrio
Altura de la línea receptora	4 m sobre los receptores primarios
Anchura del receptor	0.3 m
Tipo de absorbedor	Tubo de acero DIN 1.4541 (AISI 321) Acero inoxidable austenítico estabilizado
Fluido de trabajo	Agua
Generación de vapor	No
Presión de diseño	13 bar
Número de filas de reflectores primarios por línea receptora	22 filas
Longitud de cada módulo reflector	4 m
Anchura de los reflectores	0.5 m
Reflectividad	0,92
Relación de concentración	25

A continuación, se detallan los componentes del colector solar:

- La estructura de acero marca los límites del área donde están situados los espejos, está recubierta con pintura en polvo y sostiene tanto los espejos como el tubo de absorción y el reflector secundario.
- Los espejos reflectores son de vidrio y están unidos a un eje que es movido por un motor impulsor que se encargar del seguimiento solar. Por seguridad son ligeramente curvados elásticamente, el radio de curvatura es de entre 8,6 y 10.6 metros.
- Los motores colocados en cada fila mueven 8 espejos de cada fila, cuatro en cada lado.
- El reflector secundario envuelve al tubo captador y refleja la radiación solar que se desvía del receptor para aumentar la eficiencia óptica del sistema. También protege al receptor.
- El tubo receptor tiene una absortividad nominal de 0.94, tiene un diámetro de 70mm y una longitud de

64 m. La temperatura máxima soportada es de 200 ºC y la presión máxima en su interior de 16 bares.

- Además de los componentes anteriores, en el sistema se utilizan diversos sensores que se comunican a través de CANopen con el autómata encargado del control de la planta, estos son los siguientes:
    - Sensor solar: se utiliza para la calibración automática detectando la reflexión de los espejos primarios no centradas en el receptor.
    - Potenciómetro: cada fila de espejos cuenta con un sensor de este tipo que envía la posición actual de cada fila.
    - Sensor de temperatura: a la entrada y salida del captador se encuentra un sensor PT100 instalado para monitorizar la temperatura del fluido.

La mayoría de los componentes de la instalación se pueden ver en la Figura 2-4, esta fotografía fue tomada en una de las primeras visitas a la planta para la comprobación de su estado.

Figura 2-4.- Campo solar de la ETSI

3 M340/UNITY PRO XL

Desde la aparición de los autómatas programable la industria ha experimentado grandes avances con respecto a la solución de procesos complejos y control de las variables existentes dentro de estos procesos.

Los autómatas programables dentro de la industria son utilizados para diversas aplicaciones como pueden ser procesos simples o procesos más complejos que requieran de una alta precisión. Debido a su eficiencia y fiabilidad ha aumentado su utilización en el campo de la generación de energía eléctrica con el uso de energías renovables.

3.1 Autómatas programables

Un controlador lógico programable, más conocido por sus siglas en inglés, PLC (Programmable Logic Controller), es un computador que se utiliza en la automatización industrial para el control de procesos electromecánicos. Este computador físico está diseñado para el control de procesos secuenciales que se ejecutan en un ambiente industrial, es decir, están conectados a la maquinaria que desarrolla los procesos y controlan su trabajo. La estructura general de un autómata se puede considerar como un sistema basado en un microprocesador, siendo sus partes fundamentales la CPU (Unidad Central de Proceso), la memoria y el sistema de Entradas y Salidas (E/S). La CPU es la encargada del control interno y externo del autómata, así como la interpretación de las instrucciones del programa. Las señales de las salidas están generadas a partir de las instrucciones almacenadas en la memoria y de los datos que recibe en las entradas.

Figura 3-1.- Autómata programable Siemens S7-300 [9].

3.2 Configuración del PLC y sensores

El PLC de la Figura 3-2 que se encuentra instalado en la planta solar es un Modicom M340 de Schneider Electric. Tiene instalados 4 módulos:

- BMX P34 20302: Bus CANopen, hasta 64 esclavos y 512 direcciones de memoria.
- BMX DDI 1602: Entradas digitales, 16 entradas digitales a 24 Vcc común positivo.
- BMX DDO 6402K: Salidas digitales, 64 salidas digitales transistor común negativo.
- BMX AMI 0410: Entradas analógicas, 4 entradas analógicas U/I con separación de potencia de alta velocidad.

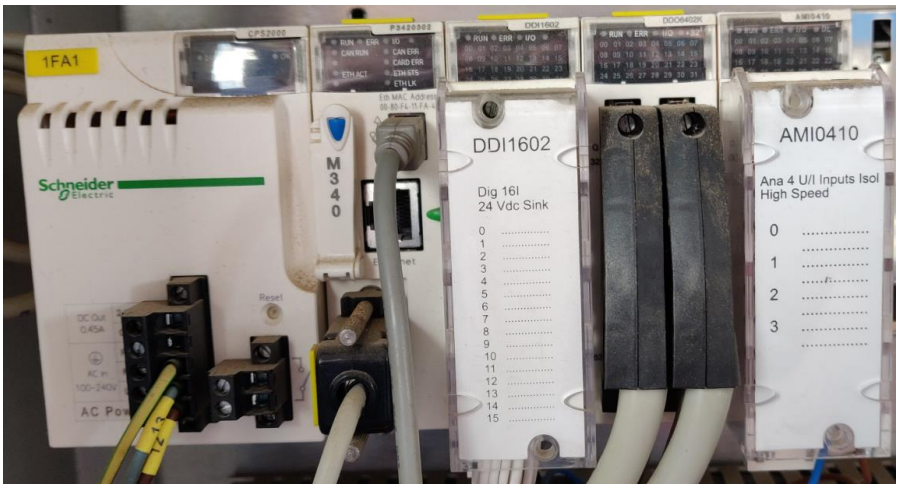

Figura 3-2.- Modicom M340 y módulos.

El campo solar cuenta con diversos sensores, entre ellos varios sensores de temperatura y varios sensores solares, de los cuales uno será el utilizado para la calibración.

Existen dos campos, cada uno de ellos tiene once filas de espejos y cada fila es movida por un servomotor. Como se comentó anteriormente, estos servomotores cuentan con un potenciómetro para conocer en todo momento la posición de los espejos, esta información es enviada desde el campo hasta el PLC a través del bus CANopen.

El módulo de entradas del autómata está vinculado a las siguientes variables:

Tabla 2.- Entradas digitales autómata M340.

ENTRADAS DIGITALES	
CANAL M340	**DESCRIPCIÓN**
%I0.1.00	Entrada 1 SAI
%I0.1.01	Sobrecalentamiento
%I0.1.02	Seta emergencia externa
%I0.1.03	Fallo cadena seguridades
%I0.1.04	Entrada 2 SAI
%I0.1.05	Detector de lluvia
%I0.1.06	Colector ON
%I0.1.07	Bomba ON

Estas variables serán utilizadas para avisar al programa principal de sucesos que pueden provocar un gran cambio en su funcionamiento.

La mayoría de las salidas del autómata serán principalmente salidas lógicas para establecer el sentido de giro de los diversos servomotores, en la siguiente tabla se puede ver la correspondencia de las salidas del módulo con sus respectivos equivalentes físicos.

Tabla 3.- Salidas digitales autómata M340.

SALIDAS DIGITALES			
CANAL M340	**DESCRIPCIÓN**	**CANAL M340**	**DESCRIPCIÓN**
%Q0.2.00	Fallo colector	%Q0.2.01	Cadena seguridades
%Q0.2.02	Apagado activado	%Q0.2.03	Reserva
%Q0.2.04	Giro horario servo 1.1	%Q0.2.05	Giro antihorario servo 1.1
%Q0.2.06	Giro horario servo 1.2	%Q0.2.07	Giro antihorario servo 1.2
%Q0.2.08	Giro horario servo 1.3	%Q0.2.09	Giro antihorario servo 1.3
%Q0.2.10	Giro horario servo 1.4	%Q0.2.11	Giro antihorario servo 1.4
%Q0.2.12	Giro horario servo 1.5	%Q0.2.13	Giro antihorario servo 1.5
%Q0.2.14	Giro horario servo 1.6	%Q0.2.15	Giro antihorario servo 1.6
%Q0.2.16	Giro horario servo 1.7	%Q0.2.17	Giro antihorario servo 1.7
%Q0.2.18	Giro horario servo 1.8	%Q0.2.19	Giro antihorario servo 1.8
%Q0.2.20	Giro horario servo 1.9	%Q0.2.21	Giro antihorario servo 1.9
%Q0.2.22	Giro horario servo 1.10	%Q0.2.23	Giro antihorario servo 1.10
%Q0.2.24	Giro horario servo 1.1	%Q0.2.25	Giro antihorario servo 1.11
%Q0.2.32	Giro horario servo 2.1	%Q0.2.33	Giro antihorario servo 2.1
%Q0.2.34	Giro horario servo 2.2	%Q0.2.35	Giro antihorario servo 2.2
%Q0.2.36	Giro horario servo 2.3	%Q0.2.37	Giro antihorario servo 2.3
%Q0.2.38	Giro horario servo 2.4	%Q0.2.39	Giro antihorario servo 2.4
%Q0.2.40	Giro horario servo 2.5	%Q0.2.41	Giro antihorario servo 2.5
%Q0.2.42	Giro horario servo 2.6	%Q0.2.43	Giro antihorario servo 2.6
%Q0.2.44	Giro horario servo 2.7	%Q0.2.45	Giro antihorario servo 2.7
%Q0.2.46	Giro horario servo 2.8	%Q0.2.47	Giro antihorario servo 2.8
%Q0.2.48	Giro horario servo 2.9	%Q0.2.49	Giro antihorario servo 2.9
%Q0.2.50	Giro horario servo 2.10	%Q0.2.51	Giro antihorario servo 2.10
%Q0.2.52	Giro horario servo 2.11	%Q0.2.53	Giro antihorario servo 2.11

3.3 Funcionamiento de la planta

El funcionamiento completo de la planta se puede describir con el diagrama de flujos de la Figura 3-3. En ella podemos encontrar un primer estado de reposo donde se espera hasta que se le índice mediante una señal externa el modo de funcionamiento. Dentro del modo Manual hay varias posibilidades como puede ser la etapa de Limpieza donde se colocan todos los servomotores en una posición predeterminada para facilitar la limpieza de los espejos de cada fila. En el estado Set se colocarán los espejos en las posiciones indicadas de forma externa. En la etapa de mantenimiento se colocarán los espejos en una posición vertical para facilitar el acceso al personal para comprobar el funcionamiento de los potenciómetros y servomotores fácilmente. La calibración se realiza de forma manual, los datos se almacenan y se utilizan como referencia para que la etapa de seguimiento funcione correctamente, esta última etapa es la encargada de que los espejos estén orientados hacia el tubo captador, tendrá en cuenta los cálculos internos para el desenfoque de algunos espejos y el error en la calibración de cada una de las filas por separado.

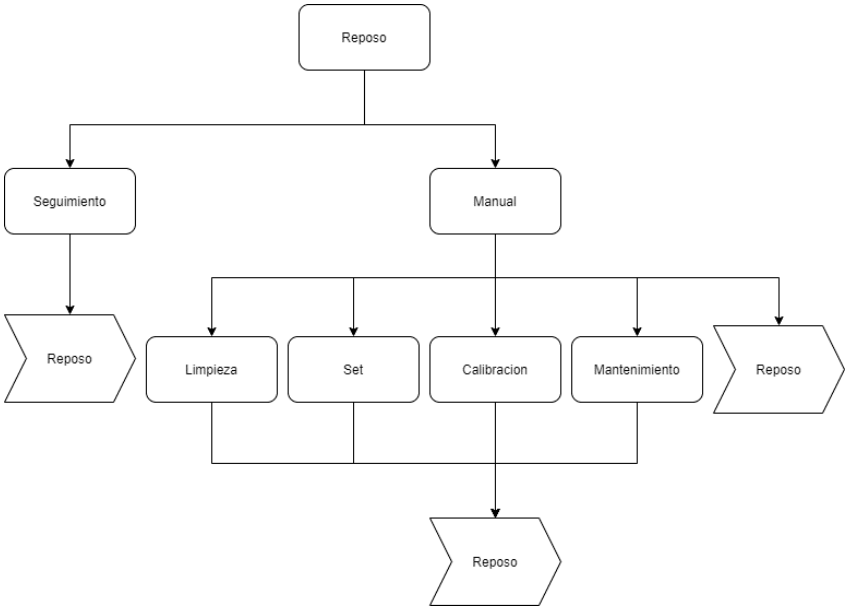

Figura 3-3.- Diagrama de flujos del funcionamiento de la planta.

3.4 Programa de calibración

El proceso de calibración de una planta solar es muy importante, ya que el resto del control dependerá de este paso. Como se ha mencionado anteriormente, este proceso se hace de forma manual pero no es eficiente ni tan exacto como si se realizase de forma automática. Por ello se ha desarrollado un programa que se encargue de este paso y que no necesite de un operario para su funcionamiento, esto permitiría ejecutar la calibración de forma más asidua, en caso de reinicio y pérdida de los valores del sistema. Solo habría que lanzar este modo de funcionamiento sin necesidad de subir a la planta y hacer la calibración manual. El desarrollo del programa de calibración se realizará en el software Unity Pro XL, los lenguajes utilizados serán ladder, texto estructurado y grafcet, este último será el encargado del control de las etapas y transiciones del programa.

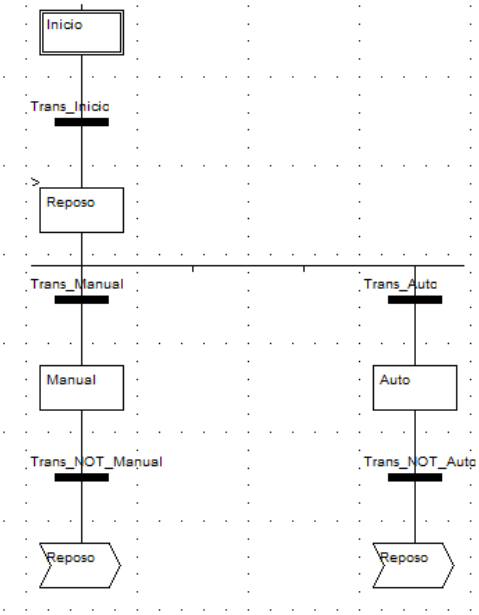

Figura 3-4.- Grafcet principal.

El primer paso era estructurar el programa en un modo de funcionamiento automático y uno manual, este último sería para controlar los servomotores individualmente, Figura 3-5. Antes de proceder con la calibración había que probar que todos los servomotores de la planta funcionasen, para ello en el modo manual se programó una etapa individual para el funcionamiento de cada servomotor.

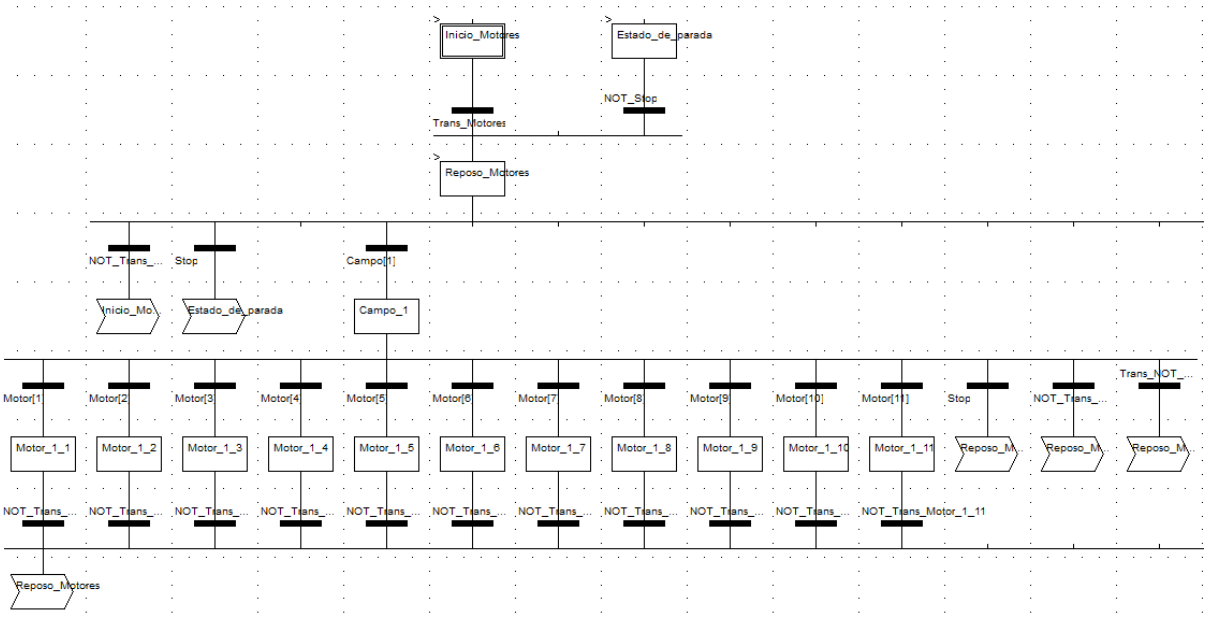

Figura 3-5.- Grafcet movimiento individual de los motores.

El funcionamiento de los servomotores depende de dos señales, una de ellas indica sentido de giro horario y la otra indicada sentido de giro antihorario, conociendo la posición actual y la posición objetivo, se puede determinar el sentido de giro deseado para cada servomotor individualmente.

```
Posicion_actual:=INT_TO_REAL(PosicionCAN_1_1);

(*Se comprueba si queremos girar en sentido horario o antihorario*)
IF (Posicion_obj>(Posicion_actual+0.05) AND Stop=0) THEN
        Giro_pos_campo_1[1] := TRUE;
        Giro_neg_campo_1[1] := FALSE;

ELSIF (Posicion_obj<(Posicion_actual-0.05)  AND Stop=0) THEN
        Giro_pos_campo_1[1] := FALSE;
        Giro_neg_campo_1[1] := TRUE;

ELSE
        Giro_pos_campo_1[1] := FALSE;
        Giro_neg_campo_1[1] := FALSE;
END IF;
```

Figura 3-6.- Movimiento motores Unity Pro XL.

Como la planta solar es un sistema real en el que un fallo puede acarrear un coste, se preparó una simulación del funcionamiento de los motores dentro del propio programa para comprobar que cada sección preparada para el movimiento de estos, funcionaba correctamente. Se utilizó un temporizador para aumentar o disminuir el ángulo de los servomotores en función de si la señal recibida indicaba sentido de giro horario o antihorario.

```
IF (sim=1) THEN(*Temporizador*)
        TON_1(PT := t#0.1s, Q=>Movimiento);

        (*--------------------CAMPO 1--------------------*)

        (*Motor 1*)
        IF (Giro_neg_campo_1[1]=1) THEN
                TON_1( IN := 1);
        END_IF;

        IF (Giro_neg_campo_1[1]=1 AND Movimiento) THEN
                PosicionCAN_1_1:=PosicionCAN_1_1-1;
                TON_1( IN := 0);
        END_IF;

        IF (Giro_pos_campo_1[1]=1) THEN
                TON_1( IN := 1);
        END_IF;

        IF (Giro_pos_campo_1[1]=1 AND Movimiento) THEN
                PosicionCAN_1_1:=PosicionCAN_1_1+1;
                TON_1( IN := 0);
        END_IF;
```

Figura 3-7.- Simulación funcionamiento servomotor Unity Pro XL.

El objetivo de la calibración es conocer el ángulo correspondiente de cada espejo en el que reflejan la máxima radiación al sensor situado encima del tubo por donde circula el fluido. Para ello se moverán los espejos en un sentido arbitrario hasta detectar que el sensor se active.

```
(*-----Descripcion de la funcion-----*)

(*-----Fila 1-----*)

IF SensorCalibracion1 = 1 THEN
        GiroPos1Campo1 := FALSE;
        GiroNeg1Campo1 := FALSE;
ELSE
        GiroPos1Campo1 := TRUE;
END_IF;
```

Figura 3-8.- Proceso calibración básico de una fila de espejos.

Luego se almacenaría el valor recibido por los potenciómetros y se utilizaría este valor como referencia de la posición óptima en la cual se refleja la máxima radiación hacia el captador. Esta primera versión cumplía con los requisitos, pero para mejorarla, se buscó una forma de obtener un ángulo más exacto. Como la medida del sensor tiene un rango de valores, cuanto más centrada esté la radiación reflejada mayor será el valor leído en el sensor, como si fuese una campana de Gauss. En la Figura 3-9 se puede observar mejor la idea expresada anteriormente, la figura es un modelado 3D de la planta realizado en Unity 3D que se comentará más adelante en el capítulo 4.

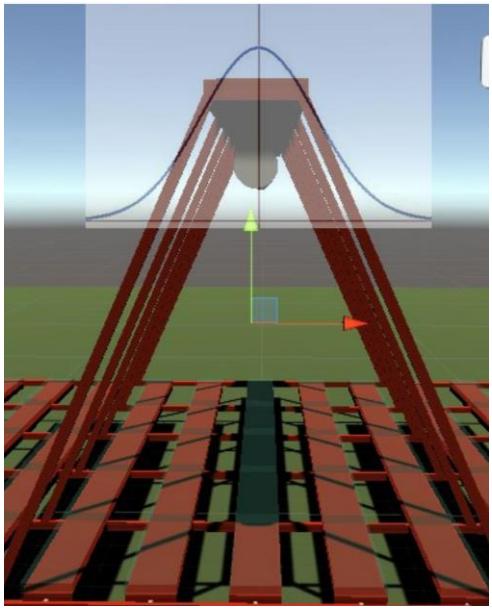

Figura 3-9.- Versión mejorada calibración.

Una vez se detecte que el valor deja de crecer, se sabrá que se ha pasado por el pico de la campana de Gauss, por ello se girará en sentido contrario hasta volver a detectar este máximo y obtener un ángulo mucho más exacto.

También se desarrolló una pantalla de operador para controlar y visualizar la planta desde el propio Unity Pro XL, en esta podríamos escoger entre los dos modos de funcionamiento, el automático que realizaría la calibración y el manual que nos permitiría seleccionar un campo y después el motor que deseásemos mover. En cualquier momento se podría pulsar el botón de 'Stop' y el autómata se iría a un estado de emergencia parando todos los procesos que estuviese haciendo y cuando se desactivase habría que empezar a escoger el modo de funcionamiento. Para el modo manual habría 5 botones que permitirían dar valores exactos a la posición deseada donde tendría que moverse el servomotor, también se contaba con un campo de entrada donde se podría introducir un valor manualmente. Por otra parte, en caso de querer simular el movimiento de los motores solo tendríamos que pulsar el botón que lo indica. En todo momento se podría observar la posición de todos los servomotores en los cuadros respectivos a cada campo y motor.

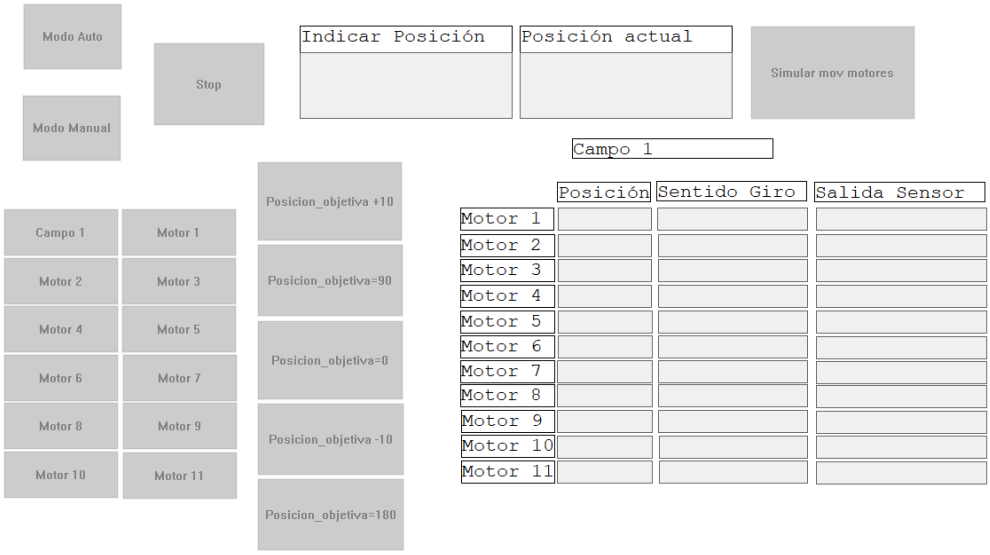

Figura 3-10.- Pantalla de operador Unity Pro XL.

3.5 Comunicaciones

Para que el autómata pueda funcionar, necesita conocer las posiciones de los servomotores, así como las lecturas de los sensores instalados en el campo solar. La planta original se comunicaba con el PLC mediante un bus CANopen.

3.5.1 Descripción CANopen

CANopen es un protocolo de comunicaciones de alto nivel, especializado en la industria, está basado en el bus CAN (ISO 11898) y recogido en la norma EN 50325-4 [10]. Este protocolo ha sido diseñado por CiA (CAN in Automation), una asociación sin ánimo de lucro constituida por usuarios y fabricantes del bus CAN, entre 1992 y 1999 [11].

El bus CAN sigue el modelo de referencia OSI, aunque suprime las capas intermedias para así aligerar la implementación, ya que los servicios ofrecidos por dichas capas no serían de utilidad para un bus industrial.

El dispositivo sigue una sencilla estructura que consta de tres unidades funcionales:

- Aplicación: relaciona las funciones del dispositivo con el proceso.
- Comunicaciones: facilita los objetos de comunicación y se encarga de la transmisión de datos a través de la red subyacente.
- Diccionario de objetos: recopilación de elementos que influyen en los objetos de comunicación, de aplicación, así como en la máquina de estados del dispositivo.

Por otro lado, en cuanto a las comunicaciones, este modelo permite la transmisión tanto síncrona como asíncrona de información. Así, existen tres tipos de relaciones en la comunicación:

- Maestro/Esclavo: un dispositivo “maestro” es el encargado de administrar la red mediante peticiones a los dispositivos “esclavos”.
- Cliente/Servidor: en este caso es el cliente el que realiza la petición al servidor.
- Productor/Consumidor: distinguimos entre:
    - Modelo push: este modelo se utiliza para la transmisión de datos a alta velocidad, ya que el consumidor no confirma la recepción de los datos enviados por el productor.
    - Modelo pull: en este caso sí hay confirmación, por lo que la velocidad es menor pero su fiabilidad es mayor.

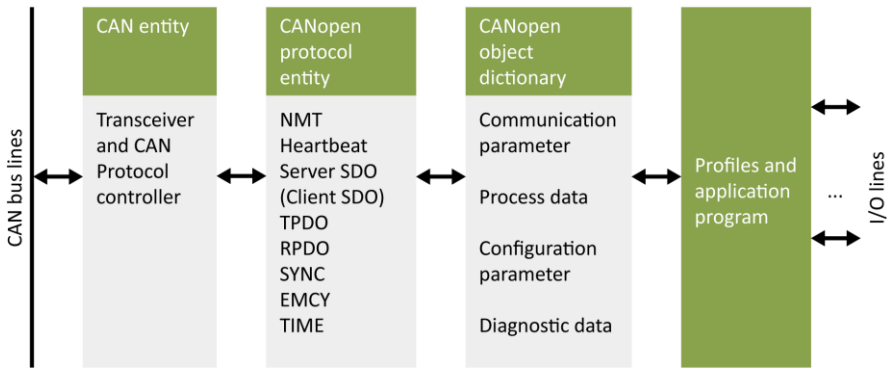

Figura 3-11.- Modelo dispositivo CANopen.

3.5.2 Estructura CANopen

La comunicación estaba compuesta por 39 nodos, separados en 4 segmentos: PCAN-8, PCAN-9, PCAN-10 y PCAN-11. Estos segmentos están conectados mediante unos repetidores CAN-CR200 para comunicarse mediante una única entrada con el módulo P34 20302.

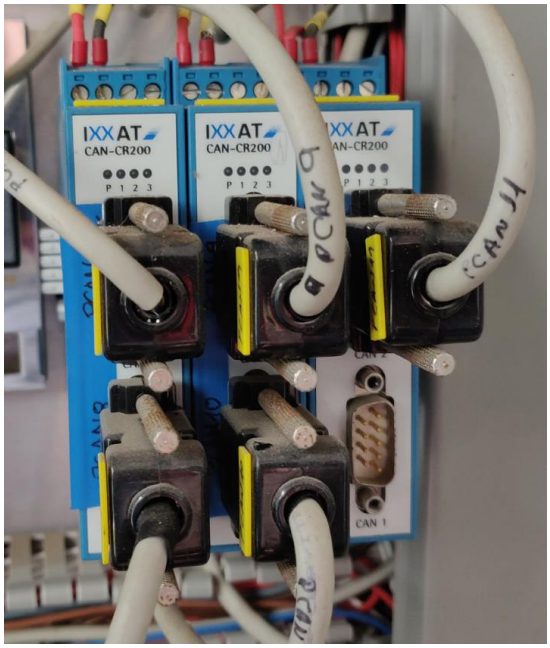

Figura 3-12.- Repetidores CAN-CR200 para bus CANopen.

PCAN-8 y PCAN-9, comunican los 22 potenciómetros situados en los motores de los espejos mientras que los otros dos segmentos conectan diversos sensores situados en el campo con el autómata. Quedarían estructurados de la siguiente forma:

Tabla 4.- Disposición bus CANopen.

Segmento CAN							
PCAN-8		**PCAN-9**		**PCAN-10**		**PCAN-11**	
Nodo ID	**Descripción**	**Nodo ID**	**Descripción**	**Nodo ID**	**Descripción**	**Nodo ID**	**Descripción**
1	Posición servo 1.1	21	Posición servo 2.1	41	Sensor solar 1.1	61	Sensor solar 2.1
2	Posición servo 1.2	22	Posición servo 2.2	42	Sensor solar 1.2	62	Sensor solar 2.2
3	Posición servo 1.3	23	Posición servo 2.3	43	Sensor solar 1.3	63	Sensor solar 2.3
4	Posición servo 1.4	24	Posición servo 2.4	44	Sensor solar 1.4	64	Sensor solar 2.4
5	Posición servo 1.5	25	Posición servo 2.5	45	Sensor Temperatura 1	65	Sensor T2 (temperatura sensor 2.1)
6	Posición servo 1.6	26	Posición servo 2.6	46	Irradiation (Sensor solar)	66	Temperatura salida del colector (T-out-collector)
7	Posición servo 1.7	27	Posición6servo 2.7	47	Temperatura cuadro (PT100)		
8	Posición servo 1.8	28	Posición servo 2.8	48	Espejos en modo rastreo (ANA INP)		
9	Posición servo 1.9	29	Posición servo 2.9				
10	Posición servo 1.10	30	Posición servo 2.10				
11	Posición servo 1.11	31	Posición servo 2.11				

Sin embargo, durante la realización de las pruebas de los motores se encontró un fallo en el funcionamiento de esta comunicación. Se probó a reiniciar varias veces el sistema, pero el error se mantenía, los detalles del error eran bastante escasos y se pueden ver en la Figura 3-13. Se investigó sobre el funcionamiento de los repetidores del bus CAN de la Figura 3-12 y se hicieron diferentes pruebas con la conexión de solo alguno de ellos, pero el error persistía. Más tarde se supo que el panel había sido cambiado de lugar y que no se había probado el correcto funcionamiento de las conexiones. Debido al Covid-19 y al estado de alarma decretado en todo el territorio nacional el edificio se cerró y no fue posible continuar con el arreglo de la planta.

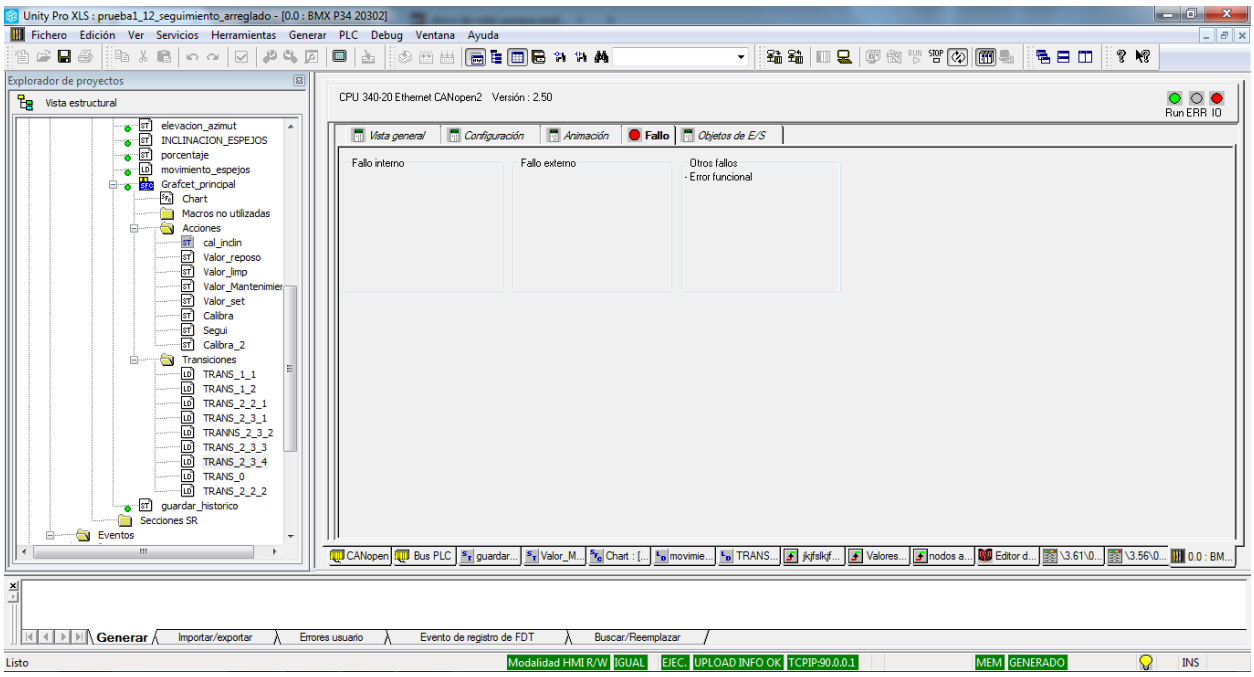

Figura 3-13.- Error funcional del módulo de comunicación CANopen.

4 SIMULADOR

Las simulaciones nos permiten interactuar con una réplica virtual de un sistema real, sin necesidad de tener que parar el funcionamiento de este o evitar la ruptura de un componente si una actualización del software no funcionase correctamente. En una planta solar, esto se transmite en la pérdida de energía durante las pruebas, por lo que si fuese posible testear previamente si una actualización del código funcionara correctamente evitaría posibles fallos. Además, se necesitarían menos pruebas reales para comprobar su correcto funcionamiento. Del mismo modo, el uso de un simulador permitiría al personal de investigación y al alumnado el acceso a la planta, de manera virtual y simultánea, para poder ensayar estrategias de control y mantenimiento sobre la misma.

4.1 Unity 3D

4.1.1 Introducción

Una vez justificada la finalidad de usar un simulador para el desarrollo del proyecto, fue necesario recurrir a un motor gráfico utilizado en el desarrollo de videojuegos para suplir las funciones del simulador al no disponer de un simulador de una planta solar tipo fresnel.

Unity 3D permite el desarrollo de videojuegos tanto en 2D como en 3D. Su versatilidad permite al usuario desarrollar proyectos para OS X, Windows o Linux, además permite exportar fácilmente los proyectos a una gran cantidad de plataformas como pueden ser PCs, Smartphones, consolas, Smart TVs o dispositivos de Realidad Virtual.

Entre los principales motivos por lo que se ha optado por utilizar este software están la existencia de una licencia gratuita para proyectos cuyo desarrollo no supere los 100.000$, como es el caso de este proyecto, y la gran versatilidad y potencia que proporciona.

4.1.2 Características técnicas

Las características técnicas que nos interesa mencionar por haber sido utilizadas en el proyecto son las siguientes.

En primer lugar, el motor gráfico de Unity 3D para Windows puede utilizar Direct3D u OpenGL, este último también es utilizado en Mac y Linux. Para Android y iOS se hace uso de OpenGL ES. Tiene soporte para mapeado de reflejos, mapeado por paralaje, mapeado de relieve, oclusión ambiental en espacio de pantalla, sombras dinámicas utilizando mapas de sombra, efectos de post-procesamiento de pantalla completa y render de texturas.

También se ha usado el soporte integrado para Nvidia, el motor de física PhysX, con soporte en tiempo real para mallas arbitrarias y sin piel, ray casts gruesos y uno de los aspectos más importantes y potentes, las capas de colisión.

Por último, el scripting se basa en Mono, la implementación de código abierto de .NET Framework. Los programadores pueden utilizar UnityScript (un lenguaje personalizado inspirado en la sintaxis ECMAScript y el más recomendado), C# o Boo (que tiene una sintaxis inspirada en Python).

4.1.3 Introduccion a Unity 3D

Para obtener Unity 3D solo tendremos que acceder a su página principal https://unity3d.com/es [12] y completar el registro. Una vez finalizado este paso solo tendremos que hacer clic en la pestaña “Productos”, “Primeros pasos” dentro de la sección “Plataforma básica”, seleccionar la pestaña Persona y ahí ya podremos proceder a la descarga.

4.1.3.1 Creación de un Proyecto

Para la creación de un nuevo proyecto solo tendremos que inicializar el programa y seleccionar la opción de New Project, aparecerán distintas plantillas entra las que se encuentra la 3D, que será la utilizada en el proyecto.

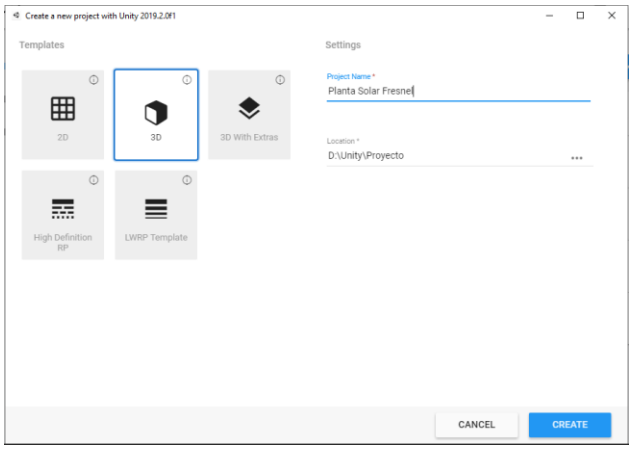

Figura 4-1.- Creación de un nuevo proyecto.

4.1.3.2 Elementos principales

Para un mejor entendimiento del programa será necesario conocer cuáles son los elementos principales de Unity 3D, además nos permitirá un mejor entendimiento del funcionamiento de este software y del resto del proyecto. Estos elementos serán nombrados posteriormente con una elevada frecuencia manteniendo siempre la nomenclatura de Unity 3D, por lo que se explicarán a continuación.

- *Game object*: principalmente es la base en la creación de contenido dentro de Unity 3D, como su traducción literal indica, son objetos del juego. A estos objetos se les pueden añadir componentes, por lo que cada game object estará definido por los componentes que tenga asignados. Además, una vez colocados en la escena podrán ser nombrados con el fin de identificarlos si hubiese dos iguales.
- *Camera*: la cámara será la encargada de renderizar todo lo que se encuentre en su campo de visión para el *Game view*. Es un *game object* imprescindible, ya que sin él no podremos ver nada fuera de la pantalla de la escena. Al comenzar una escena siempre se generará una *main camera* por defecto
- *Light*: es otro de los *game object* que se crea en cada escena, es el encargado de añadir la luz a la escena, está luz puede ser parametrizada, colocada y orientada como se desee.
- *Component*: los componentes son la parte más esencial del editor de Unity 3D, como se mencionó anteriormente, determinan como será cada *game object*. Existen muchos componentes diferentes, además, el usuario puede crear sus propios componentes.
- *Scene*: la escena se define como un nivel o pantalla dentro del juego, estará compuesta por diversos *game objects*, entre los que se encontrarán los *game objects light y camera.* Un proyecto podrá contener varias escenas y se podrá cambiar entre ellas en el momento deseado. En el juego solo se mostrará la parte de la escena que se encuentre en el campo de visión de la cámara.
- *Transform*: todos los *game objects* pertenecientes a una escena tendrán este componente. Indica la posición, rotación y escala de dicho objeto dentro de la escena. Según si el *game object* está asociado a otro *game object* (padre), el valor de estas magnitudes estará definida también respecto a las del *game object* al que está referido.
- *Scripts:* dentro de los componentes este es uno de los más potentes, pues permite programar cómo será el desarrollo del *game object* durante el juego. Cuenta con la capacidad de leer y modificar muchos parámetros de los demás componentes del propio *game object* o de otros *game objects* en la misma escena. El potencial de los scripts está limitado a la imaginación y capacidad de programar del usuario. Dentro de los lenguajes permitidos en este proyecto utilizaremos C# por su similitud con C++ y por la cantidad de cursos gratuitos que existen.

- *Tag*: como su traducción indica es una etiqueta que tiene cada *game object* y sirve para vincular varios de estos con el fin de identificarlos fácilmente en los *scripts*.
- *Gizmo*: es la representación de los ejes de un *game object* dentro de la escena, se puede actuar sobre ellos para realizar cambios en el *transform* del *game object* correspondiente.
- *Asset*: es un término de Unity 3D que se utiliza para definir cualquier elemento almacenado en la biblioteca del editor y que podrá utilizarse más tarde como plantilla.
- *Prefab*: al igual que *Asset*, los *prefabs* son un término propio de Unity 3D que hace referencia a un *game object* almacenado en la biblioteca que podrá ser instanciado en cualquier momento manteniendo los componentes y propiedades. Cuenta con la ventaja de que modificando el *prefab* se modificarán todas las instanciaciones de este, en casos de arreglar un bug solo habría que solucionarlo en el *prefab* en lugar de en los *game objects* instanciado.

4.1.3.3 Editor

El *layout* predeterminado de Unity 3D puede no ser muy intuitivo en una primera instancia, pero una vez que conoces el funcionamiento de cada pestaña, es muy cómodo de utilizar. Si el usuario desea modificarlo, el programa cuanta con varias configuraciones o sino podría modificarla a su gusto personal.

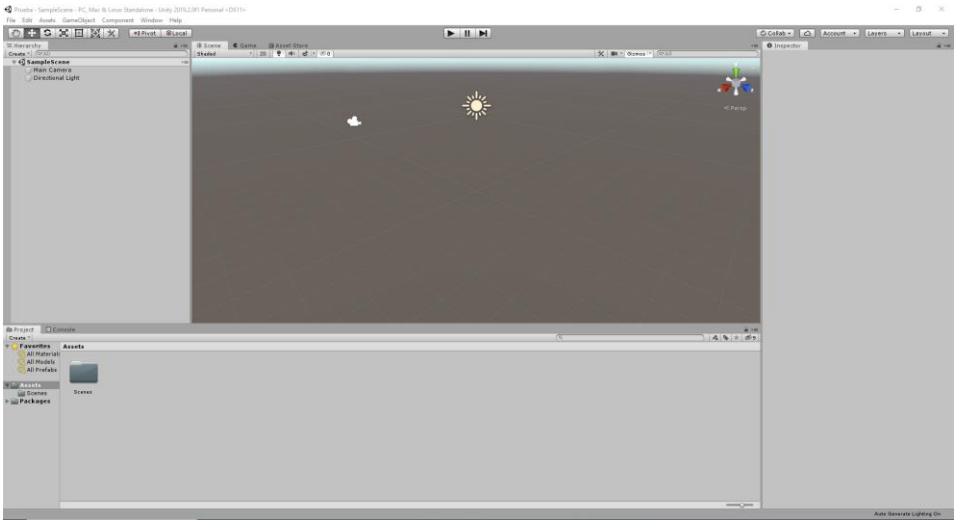

Figura 4-2.- Pantalla del editor de Unity 3D.

En la parte superior izquierda de la Figura 4-2 encontramos unos botones que nos permiten desplazar, rotar y escalar de forma muy sencilla los objetos que se encuentren dentro de la escena. En el centro de la parte superior se encuentran los botones de entrar y salir del juego, cuando se entra al juego, se cambia automáticamente de la pestaña *Scene* a la pestaña *Game*. Para modificar la escena siempre se tendrá que realizar fuera del modo juego, ya que sino cuando se abandone este modo los cambios se revertirán a como se encontraba el proyecto previo a entrar en el modo juego.

En la pestaña *Scene* será donde desarrollaremos la mayor parte del proyecto, será nuestro "mundo" en tres dimensiones. Aquí se podrán colocar *game objects* que irán apareciendo en la pestaña de la izquierda *Hierarchy.* Todos los *game objects* de la escena será mostrados en esta pestaña y se podrán asociar diferentes objetos a otro objeto que será el "padre". Como vamos a trabajar dentro del programa únicamente, la relación de aspecto de video de la pantalla *Game* se dejará en *Free Aspect*, en caso de querer exportar el proyecto a otra plataforma habría que tener esto en cuenta.

En la parte inferior de la pantalla encontramos dos pestañas, *Project* que será la biblioteca del proyecto, donde se podrán crear ficheros desde dentro del programa como desde el explorador de archivos del sistema operativo que se esté utilizando y la pestaña Console, que muestra los errores de compilación, las advertencias y los mensajes de debug que hayan sido programados en los *scripts*.

En la parte derecha encontramos la pestaña *Inspector*, aquí podremos ver todos los componentes de un *game object* y modificar algunos parámetros de estos. Desde esta pestaña se asignarán los componentes a los *game*

objects. Durante el juego se puede utilizar como debugger, ya que en los *scripts* podemos decidir si queremos que una variable se muestre en el *Inspector* y así comprobar que valores va tomando.

4.2 Estructura del proyecto en Unity 3D

El primer paso fue realizar una planta virtual funcional, es decir, filas de espejos que se pudiesen controlar desde un objeto principal llamado Controlador, al igual que sucedería en la planta real. En Unity 3D existen los 'Prefabs', son objetos que nos permiten almacenar un conjunto de objetos e instanciarlos repetidas veces en cualquier escena, modificando el prefab se modificarían todas sus instanciaciones, por lo que permite corregir cualquier error de forma simultánea. Para ello primero se creó una fila de espejos que está formada por un eje y 5 espejos, cada espejo está formado por un objeto que simula la estructura metálica situada debajo de cada espejo, otro objeto que será el cristal y por último el reflejo que emite, su normal.

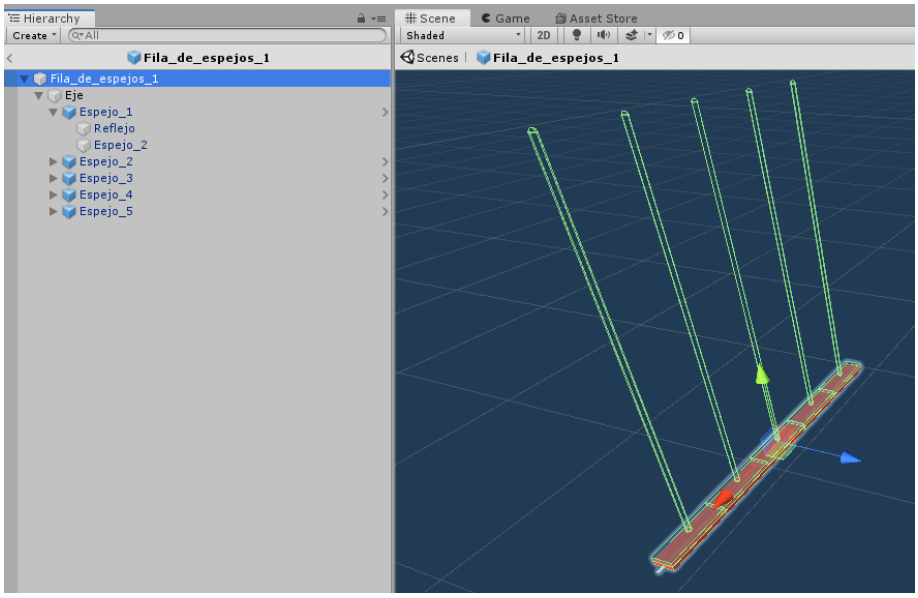

Figura 4-3.- *Prefab* fila de espejos.

Dentro de este *prefab* encontramos dos *scripts* uno para el *game object* "Fila_de_espejos_1" y otro para el *game object* "Reflejo". En caso del reflejo es importante resaltar que el componente *Capsule collider* que tiene asignado tiene activa la función *Is trigger* para activar una interrupción cuando entre en contacto con otro objeto.

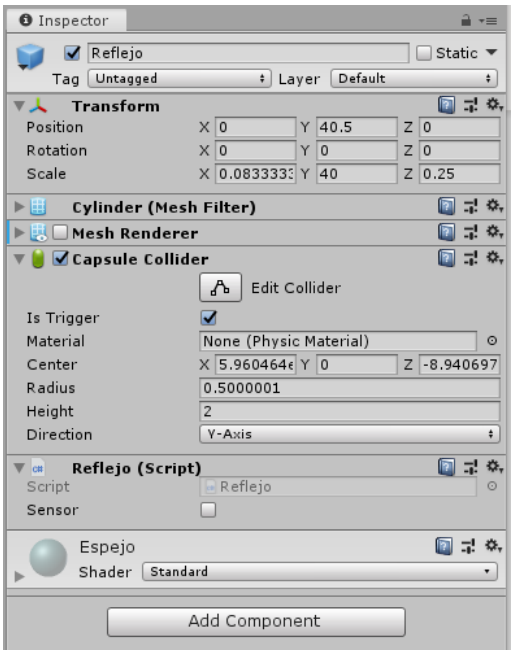

Figura 4-4.- Componentes del *game object* Reflejo.

```
6 referencias
public class Reflejo : MonoBehaviour
{
    public bool Sensor;
    0 referencias
    private void OnTriggerEnter(Collider other)
    {
        Sensor = true;
    }
    0 referencias
    private void OnTriggerStay(Collider other)
    {
        Sensor = true;
    }
    0 referencias
    private void OnTriggerExit(Collider other)
    {
        Sensor = false;
    }
}
```

Figura 4-5.- *Script Reflejo.*

Al desactivar el componente *Mesh renderer,* el objeto no se renderizará, pero seguirá estando dentro de la escena y del juego. En la Figura 4-5 podemos ver el *script* de este *game object*, la interrupción saltará según sus nombres, cuando comience la colisión, durante esta y cuando acabe. Para que la interrupción funcionase correctamente era necesario aplicarle el componente *Rigidbody* al *game object* que hará de sensor, desactivando la gravedad para el objecto como se ve en la siguiente figura:

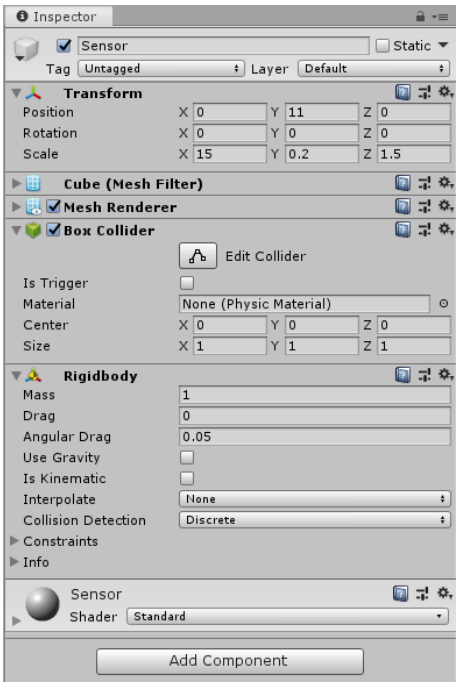

Figura 4-6.- Componentes del *game object* Sensor.

Por otro lado, el *script* que lleva asociado el *game object* "Fila_de_espejos_1" se llama Motor.cs y se encuentra en las Figura 4-7 y Figura 4-8, en la declaración de variables, estas se esconden en el *Inspector* para que sea más limpio a la hora de depurar el código. Las variables públicas pertenecientes a un objeto pueden ser leídas y modificadas por otro objeto dentro de la misma escena, sería el autómata de la planta. La función *Start* de un *script* solo se ejecutará la primera vez que aparezca el *game object* en la escena, en este caso buscarán los componentes Reflejo de cada uno de los cinco espejos que contiene el objeto.

```
public class Motor : MonoBehaviour
{
    [HideInInspector] public float rotateSpeed;
    [HideInInspector] public float Angulo;
    [HideInInspector] public bool Giro_positivo;
    [HideInInspector] public bool Giro_negativo;
    [HideInInspector] public bool Sensores;
    // Start is called before the first frame update

    [HideInInspector] public Reflejo Reflejo1, Reflejo2, Reflejo3, Reflejo4, Reflejo5;
    private GameObject Espejo;
    [HideInInspector] public GameObject Eje_de_giro;
    0 referencias
    void Start()
    {

        Eje_de_giro = gameObject.transform.Find("Eje").gameObject;

        Espejo = Eje_de_giro.transform.Find("Espejo_1").gameObject;
        Reflejo1 = Espejo.GetComponentInChildren<Reflejo>();

        Espejo = Eje_de_giro.transform.Find("Espejo_2").gameObject;
        Reflejo2 = Espejo.GetComponentInChildren<Reflejo>();

        Espejo = Eje_de_giro.transform.Find("Espejo_3").gameObject;
        Reflejo3 = Espejo.GetComponentInChildren<Reflejo>();

        Espejo = Eje_de_giro.transform.Find("Espejo_4").gameObject;
        Reflejo4 = Espejo.GetComponentInChildren<Reflejo>();

        Espejo = Eje_de_giro.transform.Find("Espejo_5").gameObject;
        Reflejo5 = Espejo.GetComponentInChildren<Reflejo>();

    }
```

Figura 4-7.- Código función *Start* de una fila de espejos.

El movimiento de los espejos se hará a través del *game object* Eje, para que sea un giro igual que en la planta solar real. El código para este movimiento y para la activación de una variable que determine que todos los espejos de la fila están reflectando la radiación sobre el sensor es el siguiente:

```
void Update()
{
    if (Giro_negativo == true && Giro_positivo == false)
    {
        Eje_de_giro.transform.Rotate(Vector3.up * rotateSpeed * Time.deltaTime);
        Angulo -= rotateSpeed*Time.deltaTime;
    }
    else if (Giro_positivo == true && Giro_negativo == false)
    {
        Eje_de_giro.transform.Rotate(Vector3.down * rotateSpeed * Time.deltaTime);
        Angulo += rotateSpeed * Time.deltaTime;
    }
    else
    {
        Eje_de_giro.transform.Rotate(Vector3.zero * rotateSpeed * Time.deltaTime);
    }
    if (Angulo > 360)
        Angulo -= 360;
    else if (Angulo < -360)
        Angulo += 360;
    else if (Angulo > 180)
        Angulo = -360 + Angulo;
    else if (Angulo < -180)
        Angulo = 360 + Angulo;
    if (Reflejo1.Sensor == true && Reflejo2.Sensor == true && Reflejo3.Sensor == true && Reflejo4.Sensor == true && Reflejo5.Sensor == true)
    {
        Sensores = true;
    }
    else
    {
        Sensores = false;
    }
}
```

Figura 4-8.- Código función *Update* de una fila de espejos.

Al ser todas las filas iguales, pero colocadas en distintas posiciones, se instanció el prefab once veces al igual que en la planta solar real y se creó un objecto llamado Controlador que sería el encargado de realizar la función del autómata programable. El controlador leería todas las posiciones de las filas de espejos, al igual que detectaría si se activa el sensor fotoeléctrico situado por encima del captador solar. Este *game object* siguió la metodología utilizada en las filas de espejos, en la función *Start* se buscarían la componente Motor de cada una de las filas de espejos y desde aquí se determinaría el sentido y velocidad de giro de cada uno de los motores independientemente.

Se le terminó de dar forma a la planta colocando objetos sin funcionalidad que formarían la estructura del captador solar de tipo fresnel.

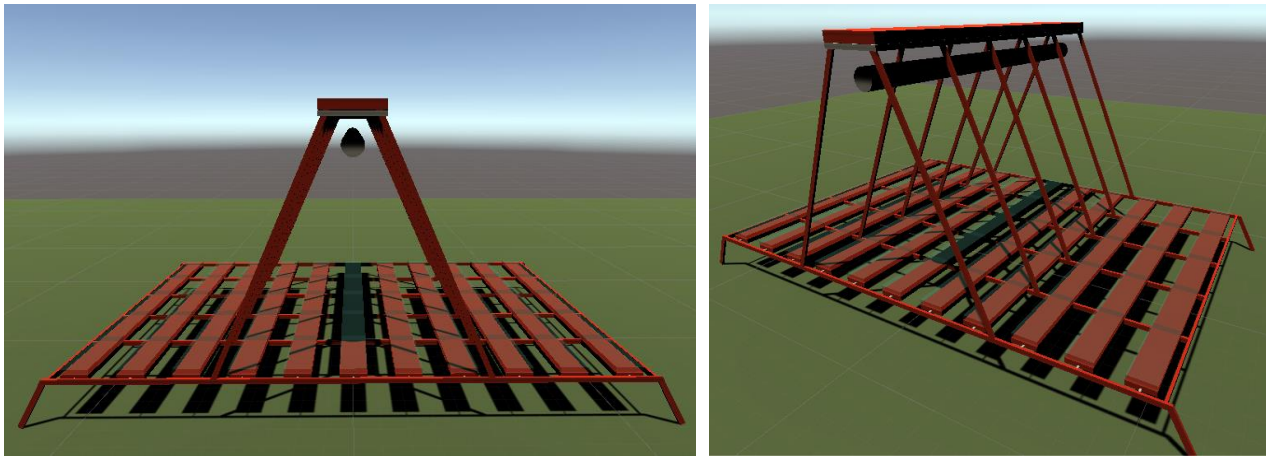

Figura 4-9.- Planta solar virtual tipo fresnel en Unity 3D.

4.3 Comunicación del Sistema de control con el simulador de la planta

Una vez se terminaron de desarrollar todos los elementos de la planta y configurado un objeto que actuaría de controlador, había que comprobar que este funcionase, esto se hizo a través de botones situados dentro del simulador, pero el objetivo principal de este proyecto era un programa de calibración eficiente para un autómata programable en Unity Pro XL. Debido a que el simulador del PLC de Unity Pro XL, acepta comunicaciones mediante el protocolo Modbus sobre TCP-IP, se procedió a desarrollar un programa en el simulador de la planta de Unity 3D, para comunicar con el mismo.

Modbus es un protocolo de comunicación de solicitud-respuesta implementado, utilizando una relación maestro-esclavo.

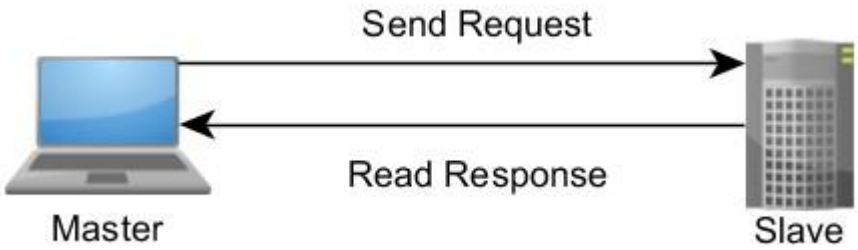

Figura 4-10.- Protocolo Modbus [13].

El protocolo Modbus sobre TCP/IP es uno de los más usados en la industria, debido a su robustez y a los más de 40 años que lleva utilizándose y obteniendo buenos resultados. Esta comunicación se puede simplificar en 4 sencillos pasos:

1. El cliente o maestro realiza una petición.
2. El servido o esclavo comunica al cliente que ha recibido su petición.
3. El servidor envía la respuesta a la petición.
4. El cliente envía una confirmación de que la respuesta a su petición ha sido recibida.

Este protocolo está situado en los niveles 1,2 y 7 del modelo OSI y para su comunicación los dispositivos colocados en la red necesitan de un socket. por tanto, la comunicación quedaría definida por una dirección IP y un puerto para cada miembro de la red, además de un protocolo común.

Puesto que la programación de un protocolo Modbus sobre TCP/IP en C# se escapa de los objetivos principales de este proyecto, se optó por utilizar una herramienta de código abierto llamada UModBusTCP. Esta herramienta contaba con diversas funciones que permitirían establecer y gestionar una conexión ModBus entre Unity Pro XL (servidor) y Unity 3D (cliente).

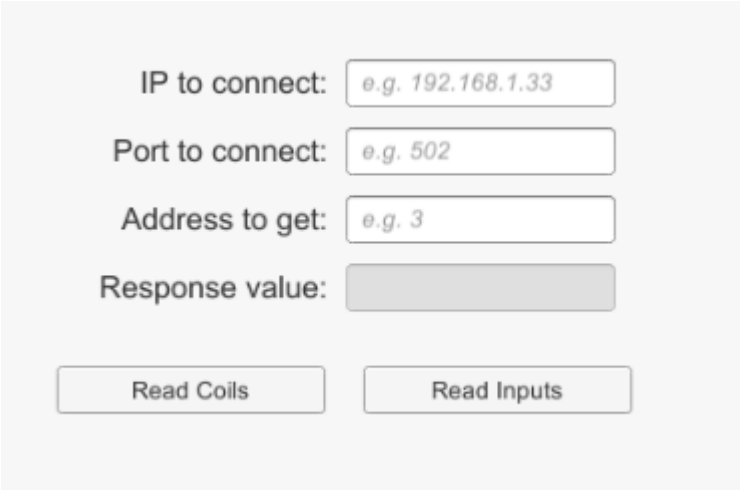

Figura 4-11.- Ejemplo herramienta UModBusTCP [14].

Junto a la herramienta encontramos un ejemplo de su uso que permitía la lectura de una bobina o de un registro, para comprobar su funcionamiento se utilizó un simulador de servidor para conexión Modbus como el que se muestra en la Figura 4-12.

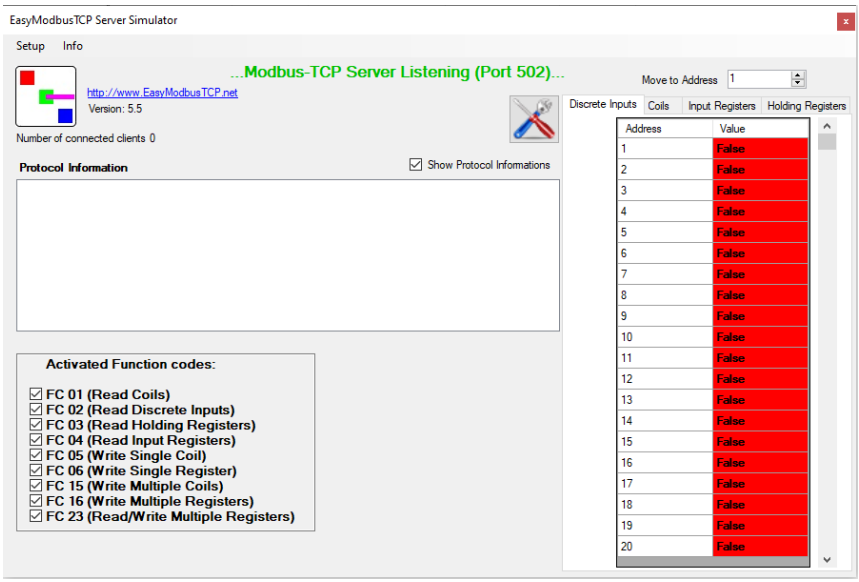

Figura 4-12.- Servidor Modbus sobre TCP/IP [15].

Las funciones de lectura de bobinas y lectura de registros funcionaban perfectamente, pero también era necesario escribir en los registros o bobinas. El resto de las funciones se encontraban definidas en la herramienta, pero no se conocía nada sobre su funcionamiento. Al igual que se utilizó un simulador de servidor también se utilizó un simulador de cliente para conocer la estructura de los mensajes y el funcionamiento de las demás funciones.

Como se vio que el ejemplo propuesto por el autor permitía la comunicación con un simulador de servidor se pasó a la implementación de esta comunicación en el proyecto. Se creó un nuevo objeto dentro de la escena que sería el encargado de la comunicación esto permitiría que un solo objeto fuese el enlazante entre ambos programas y como se mencionó antes, se podía acceder a las variables públicas de otro objeto, en este caso del Controlador.

Para la comunicación primero había que crear las variables del tipo Socket y sus respectivos buffers de tipo byte, según si la comunicación fuese a ser síncrona o asíncrona:

```
//Sockets
Socket m_oAsyncTCPSocket;
byte[] m_bAsyncTCPSocketBuffer = new byte[2048];

Socket m_oSyncTCPSocket;
byte[] m_bSyncTCPSocketBuffer = new byte[2048];
```

Figura 4-13.- Variables tipo Socket para la comunicación ModBusTCP/IP.

Una vez generados los sockets ya sería posible la conexión entre cliente y servidor, para ello se utilizarían las funciones de inicialización y finalización de la conexión que se muestra en la siguiente figura:

```
//------------------------------------------------------------------------
/// <summary>Start connection to slave.</summary>
/// <param name="_sIp">IP adress of modbus slave.</param>
/// <param name="_usPort">Port number of modbus slave. Usually port 502 is used.</param>
7 referencias
public void Connect(string _sIp, ushort _usPort, CONNECTION_MODE _eConnectionMode = CONNECTION_MODE.LINEAR)
                    //--------------------------------------------
                    /// <summary>Destroy master instance</summary>
                    2 referencias
                    public void Dispose() {
```

Figura 4-14.- Funciones de conexión y desconexión de la comunicación ModBusTCP/IP.

Estas funciones serían llamadas para comenzar y finalizar la conexión mediante unos botones cuando fuese necesario.

4.4 Primera versión

Se creó una pequeña interfaz utilizando el editor de Unity, de forma que cuando se conectase con el servidor utilizando las funciones anteriormente mencionadas.

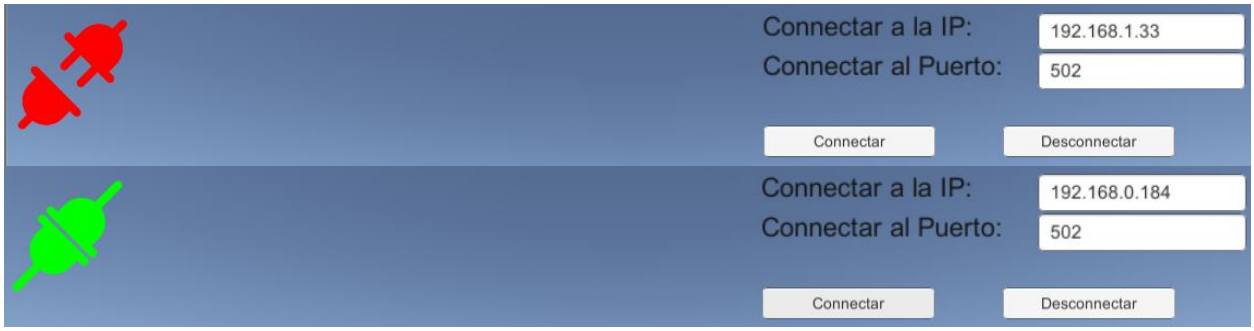

Figura 4-15.- Interfaz de la conexión sobre ModBusTCP/IP.

Después se procedió a escribir valores en el servidor virtual. La escritura de las bobinas no funcionaba correctamente y además se encontró un grave problema en la comunicación, al ser una comunicación de solicitud-respuesta esta no era suficientemente rápida para efectuarse de forma independiente para cada variable. Como la escritura de números enteros en registros individualmente, sí se consiguió que funcionase, se modificó la estructura inicial pensada para la comunicación de forma que la comunicación se realizaría con números enteros y el objecto encargado de esta, sería el encargado de traducir estos valores a las señales requeridas por el controlador. De forma que solo existiría un registro para el sentido de giro de cada motor, 0 significaría parado, 1 sentido de giro positivo y 2 sentido de giro negativo. Al tener que leer un registro menos por cada motor esto ayudo a la velocidad de la comunicación, pero aun así no seguía siendo suficiente para un intento de calibración. El siguiente paso fue intentar hacer una sola llamada de lectura de múltiples registros y una sola llamada de escritura en múltiples registros. Para ello era necesario almacenar todas las variables en vectores, esto no supuso mucha complicación en el código, pero para las funciones de escritura era necesario convertir los valores de los números enteros y las variables booleanas en bytes, esas funciones ya se encontraban en el ejemplo propuesto, pero para que funcionases con vectores hubo que realizar unos cambios que se pueden ver en la Figura 4-16.

```
public static byte[] GetBytesOfIntVector(int[] vector)
{
    byte[] b = new byte[vector.Length * 2];
        for (int i = 0; i < vector.Length; i++)
    {
        Array.Copy(UModbusTCPHelpers.GetBytesOfInt(vector[i]), 0, b, i * 2, 2);

    }
    return b;
}
0 referencias
public static byte[] GetBytesOfBoolVector(bool[] vector)
{
    byte[] b = new byte[vector.Length * 2];
        for (int i = 0; i < vector.Length; i++)
    {
        Array.Copy(UModbusTCPHelpers.GetBytesOfBool(vector[i]), 0, b, i * 2, 2);

    }
    return b;
}
```

Figura 4-16.- Funciones para convertir vectores de *Int* o *bool* en vectores de bytes.

Finalmente se decidió desistir de la escritura de bobinas ya que se consiguió escribir en más de un registro de números enteros de forma simultánea, por lo que se convirtieron los valores booleanos de los sensores en números enteros y se unieron las 22 variables de la comunicación en un solo vector, las 11 primeras posiciones serían para las posiciones de los servomotores y las siguientes 11 para las salidas de los sensores. A pesar de haber simplificado mucho la comunicación, el servidor virtual saturaba por el número de peticiones, por lo que se dividió la comunicación en dos partes, una primera llamada para la lectura y luego una llamada para la escritura, como se puede ver la Figura 4-17.

```
if (m_oUModbusTCP.connected)
{
    //Convertir los arrays a arrays de bytes
    bDatos = GetBytesOfIntVector(Datos);
    Estado.sprite = Conectado;
    if (count % 50 == 25)
    {
            ReadRegister(1,11);
    }
    if (count%50==49)
    {
        if (Sensores.Length == 1)
            m_oUModbusTCP.WriteSingleRegister(5, 1, 0, bDatos);
        else
            m_oUModbusTCP.WriteMultipleRegister(16, 1, 11, bDatos);
        count = 0;
    }
    count += 1;
}
else
{
    Estado.sprite = Desconectado;
}
```

Figura 4-17.- Lectura y escritura de registros a través de ModBusTCP desde Unity 3D.

Esto se encontraba dentro de la función *Update* que en Unity 3D se ejecuta en cada fotograma, como se realizaban muchas peticiones por segundo el servidor formaba una cola con las peticiones y se iba acumulando un retraso. Se optó por realizar una cuenta de los fotogramas y alternar una llamada de lectura con una de escritura cada 25[1] fotogramas. De este modo no se formaban colas y la comunicación era bastante fluida, al no ser una comunicación en tiempo real se disminuyó la velocidad de los motores para que no se perdiese nada de información en la comunicación, esto ralentizó el proceso de calibración, por lo que se mantuvo utilizar un sensor por cada fila de espejos para las pruebas, mientras que en la planta real solo existe un sensor solar para la calibración.

Por otro lado, desde Unity Pro XL había que declarar a que registros correspondían las variables que tenían un papel en la comunicación. Los registros quedarían del siguiente modo:

Tabla 5.- Registros comunicación ModBusTCP.

Registro	dirección de memoria	Descripción
1-11	%MW0-%MW10	Sentido de giro servomotores
12-22	%MW11-%MW21	Posiciones servomotores
23-33	%MW22-%MW32	Sensores de calibración

4.5 Segunda version

Esta calibración dependía de la velocidad de la comunicación para ser exacta, como mejora se propuso modificar el funcionamiento del sensor en Unity 3D de forma que la salida en lugar de ser 1 o 0 siguiese una distribución gaussiana, tal como ocurriría en la planta solar real. Para ello había que hacer una reestructuración del funcionamiento de la planta solar ya que el sensor dejaría de ser una salida booleana y habría que dividirlo en dos partes y utilizando las diferentes interrupciones proporcionas por las colisiones incrementar el valor de la salida mientras durante la colisión con la primera mitad del sensor y disminuirlo durante la colisión con la segunda mitad del sensor. De esta forma el punto donde la salida del sensor fuese máxima sería la unión entre las dos mitades del sensor. Debido a la complejidad de devolver como salida del sensor una distribución gaussiana, calculada a partir de las colisiones definidas por Unity 3D, se ha optado por probar dos

[1] Este valor de obtuvo de forma iterativa comprobando que el tiempo entre las llamadas no se acumulaba.

simplificaciones. En la primera de ellas, se dividió el sensor en dos partes, después, se linealizó la salida del sensor de tal forma que creciese por unidades mientras se estuviese moviendo el reflejo dentro del sensor y cuando cambiase a la segunda parte del sensor empezase a decrecer de la misma forma como se puede ver en la Figura 4-18.

```
public int Sensor = 0;
public bool StartSensor = false;
0 referencias
private void OnTriggerEnter(Collider other)
{
    if (Sensor == 0){
        Sensor++;
        StartSensor = true;
    }
    else if (Sensor > 0)
    {
        Sensor--;
        if (Sensor < 0)
            Sensor = 0;
        StartSensor = false;
    }
}
0 referencias
private void OnTriggerStay(Collider other)
{
    if (StartSensor == true && (Motor.Giro_negativo == true || Motor.Giro_positivo == true))
    {
        Sensor++;
    }
    else if (StartSensor == false && (Motor.Giro_negativo == true || Motor.Giro_positivo == true))
    {
        Sensor--;
        if (Sensor < 0)
            Sensor = 0;
    }
}
0 referencias
private void OnTriggerExit(Collider other)
{
    if (StartSensor == true)
        StartSensor = false;
}
```

Figura 4-18.- Segunda versión del sensor de calibración.

Esta versión obtuvo resultados, pero no eran los esperados, ocurrían dos problemas, la comunicación no era en tiempo real, perdiéndose información de la salida de los sensores, por lo que en Unity Pro XL hubo que poner unos márgenes de error[2] a la hora de determinar si se encontraba en el punto máximo, y que si cuando el juego comenzaba un espejo se encontraba apuntando al sensor, este daría la vuelta completa y no calibraría correctamente.

```
IF (SensorCalibracion1 > 0 OR Sensor1Calibrado) THEN
        IF Sensor1Calibrado THEN
                GiroNeg1Campo1 := FALSE;
                GiroPos1Campo1 := FALSE;
        ELSIF SensorCalibracion1 > SensorCalibracion1Last THEN
                SensorCalibracion1Last := SensorCalibracion1;
        ELSIF (SensorCalibracion1 > (SensorCalibracion1Last-5)) THEN
                Sensor1Calibrado := TRUE;
        ElSIF (SensorCalibracion1 < (SensorCalibracion1Last-20)) THEN
                GiroPos1Campo1 := TRUE;
                GiroNeg1Campo1 := FALSE;
        END_IF;

ELSE
        IF Sensor1Calibrado THEN
                GiroNeg1Campo1 := FALSE;
                GiroPos1Campo1 := FALSE;
        ELSE
                GiroNeg1Campo1 := TRUE;
        END_IF;
END_IF;
```

Figura 4-19.- Segunda versión del programa de calibración.

En un caso donde la comunicación fuese en tiempo real y el sensor funcionase como en la realidad, en lugar de como se programó en Unity 3D, los resultados obtenidos deberían ser mucho más exactos que los obtenidos

[2] Los valores 5 y 20 ha sido tomados mediante un proceso iterativo.

con la primera versión. Aún no satisfechos con los resultados, se decidió continuar con la programación del sensor en Unity 3D. Esta vez se descompuso el sensor en siete partes, entregando un valor diferente a la salida en cada una de ellas. Cuando el objeto del reflejo colisionaba con una parte del sensor lo hacía simultáneamente con otras, por lo que había que diferenciar de alguna forma cuál era cada una. Cuando se activa alguna de las interrupciones asignadas a las colisiones estas reciben como parámetro la colisión con el objecto en cuestión. A partir de esta colisión se puede determinar con que objeto está colisionando. Era importante saber con cuantos sensores estaba colisionando actualmente, como ese dato no se podía obtener del *game object*, se programó una cuenta que aumentase y disminuyese en uno cada vez que entraba o abandonaba una colisión respectivamente. Por último, había que indicar un rango de valores a la salida del sensor, como se dividió en siete partes los valores asignados fueron: 10,30,50,70,50,30,10[3], de esta forma el máximo se encontraría en el centro.

```
private void OnTriggerEnter(Collider col)
{
    countCollisions++;
    if (col.gameObject.name == "Sensor1"){
        Sensors[0] = true;        }
    else if(col.gameObject.name == "Sensor2"){
        Sensors[1] = true;        }
    else if (col.gameObject.name == "Sensor3"){
        Sensors[2] = true;        }
    else if (col.gameObject.name == "Sensor4"){
        Sensors[3] = true;        }
    else if (col.gameObject.name == "Sensor5"){
        Sensors[4] = true;        }
    else if (col.gameObject.name == "Sensor6"){
        Sensors[5] = true;        }
    else if (col.gameObject.name == "Sensor7"){
        Sensors[6] = true;        }
}
0 referencias
private void OnTriggerStay(Collider col)
{
    if (Sensors[3] == true){
        Sensor = 70;        }
    else if (Sensors[2] = true || Sensors[4] == true){
        Sensor = 50;        }
    else if (Sensors[1] = true || Sensors[5] == true){
        Sensor = 30;        }
    else if (Sensors[0] = true || Sensors[6] == true){
        Sensor = 30;        }
}
```

Figura 4-20.- Segunda versión mejorada del sensor en Unity 3D 1.

Como se puede ver en la figura anterior, se le dio prioridad a la colisión con los sensores centrales, ya que sino el programa de Unity Pro XL no funcionaría correctamente, en la realidad el funcionamiento sería similar al programado en Unity 3D, salvo que el rango de valores obtenidos sería lineal en lugar de escalonado, pero esto no influiría rendimiento del programa.

[3] Valores escogidos de forma arbitraria, podrían cambiarse, pero manteniendo el formato piramidal.

```
private void OnTriggerExit(Collider col)
{
    countCollisions--;
    if (col.gameObject.name == "Sensor1"){
        Sensors[0] = false;        }
    else if (col.gameObject.name == "Sensor2"){
        Sensors[1] = false;        }
    else if (col.gameObject.name == "Sensor3"){
        Sensors[2] = false;        }
    else if (col.gameObject.name == "Sensor4"){
        Sensors[3] = false;        }
    else if (col.gameObject.name == "Sensor5"){
        Sensors[4] = false;        }
    else if (col.gameObject.name == "Sensor6"){
        Sensors[5] = false;        }
    else if (col.gameObject.name == "Sensor7"){
        Sensors[6] = false;        }
    if (col.gameObject.name == "Sensor1" && countCollisions == 0){
        Sensor = 0;        }
    else if (col.gameObject.name == "Sensor7" && countCollisions == 0){
        Sensor = 0;        }
}
```

Figura 4-21.- Segunda versión mejorada del sensor en Unity 3D 2.

Como se mencionó anteriormente, era importante saber cuántas colisiones estaban ocurriendo en el momento del abandono de una colisión ya que en los casos de la primera y última parte del sensor la salida de este tendría que ponerse de nuevo a cero. También se realizaron cambios en el programa de Unity Pro XL sobre la calibración para que funcionase correctamente con el nuevo sensor.

```
IF (SensorCalibracion1 > 0 OR Sensor1Calibrado) THEN
        IF Sensor1Calibrado THEN
                GiroNeg1Campo1 := FALSE;
                GiroPos1Campo1 := FALSE;
        ELSIF SensorCalibracion1 > SensorCalibracion1Last THEN
                SensorCalibracion1Last := SensorCalibracion1;
        ELSIF (SensorCalibracion1 > (SensorCalibracion1Last-20) AND SensorCalibracion1MaxReached) THEN
                Sensor1Calibrado := TRUE;
        ElSIF (SensorCalibracion1 < (SensorCalibracion1Last)) THEN
                SensorCalibracion1MaxReached := TRUE;
                GiroPos1Campo1 := TRUE;
                GiroNeg1Campo1 := FALSE;
        END_IF;

ELSE
        IF Sensor1Calibrado THEN
                GiroNeg1Campo1 := FALSE;
                GiroPos1Campo1 := FALSE;
        ELSE
                GiroNeg1Campo1 := TRUE;
        END_IF;
END_IF;
```

Figura 4-22.- Segunda versión mejorada del programa de calibración.

Se introdujo una variable nueva para confirmar que se había alcanzado el máximo en la salida del sensor y para la planta real lo único que habría que modificar sería el valor 20[4] por un valor que permitiese asegurar que ni el ruido de la salida del sensor ni la velocidad de la comunicación estropeasen el funcionamiento.

[4] El valor 20 depende de los saltos especificados entre las partes de los sensores del simulador, si se cambiase el salto a 40, el máximo valor admitido sería 40, importante tenerlo en cuenta en futuras modificaciones.

5 RESULTADOS FINALES

Para probar el funcionamiento de la planta solar virtual, solo era necesario entrar en el modo juego de la planta y forzar las salidas de los motores, sin embargo, en el caso de Unity Pro XL, había que realizar la comunicación entre ambos programas, esto llevó tiempos de espera entre pruebas, ya que había que compilar y subir el proyecto al PLC virtual cada vez que se cambiaba algo del código. Para ver los resultados de la calibración se habilitó una columna más en la pantalla del operador donde se vería la salida del sensor utilizado para la calibración de cada una de las filas de espejos. Para una mejor depuración del funcionamiento de la calibración se mostró en la pantalla de operador los sentidos de giros de cada uno de los motores.

Para hacer funcionar el programa de calibración en la planta virtual, los pasos a seguir serían los siguientes:

1. Iniciar el juego en Unity 3D.
2. Transferir y ejecutar el proyecto al PLC virtual en Unity Pro XL en modo simulación.
3. En el juego indicar la dirección IP, en este caso "127.0.0.1" y pulsar el botón conectar.
4. Una vez se haya puesto verde la conexión, en la pantalla del operador pulsar el Modo Auto.

Para estar seguros de que el proceso ha comenzado solo tendríamos que irnos a la pantalla de operador y comprobar que empezamos a recibir datos de la planta como se muestra en la Figura 5-1.

	Campo 1		
	Posición	Sentido de giro	Salida del sensor
Motor 1	-2	2	0
Motor 2	-2	2	0
Motor 3	-2	2	0
Motor 4	-2	2	0
Motor 5	-2	2	0
Motor 6	0	0	70
Motor 7	2	1	0
Motor 8	2	1	0
Motor 9	2	1	0
Motor 10	2	1	0
Motor 11	2	1	0

Figura 5-1.- Comienzo proceso de calibración.

5.1 Pruebas con imágenes

Para poder mostrar el funcionamiento de ambas versiones del programa de calibración se renderizará el primer reflejo de cada una de las filas de espejos, de modo que se podrá ver la posición final de los reflejos.

Con la primera calibración en sentido de giro inicial en busca de los sensores influía en los resultados obtenidos ya que una vez hubiese colisionado con el sensor, acabaría en programa, se puede ver que los resultados obtenidos, aunque concuerdan con lo programado no son válidos para una buena calibración.

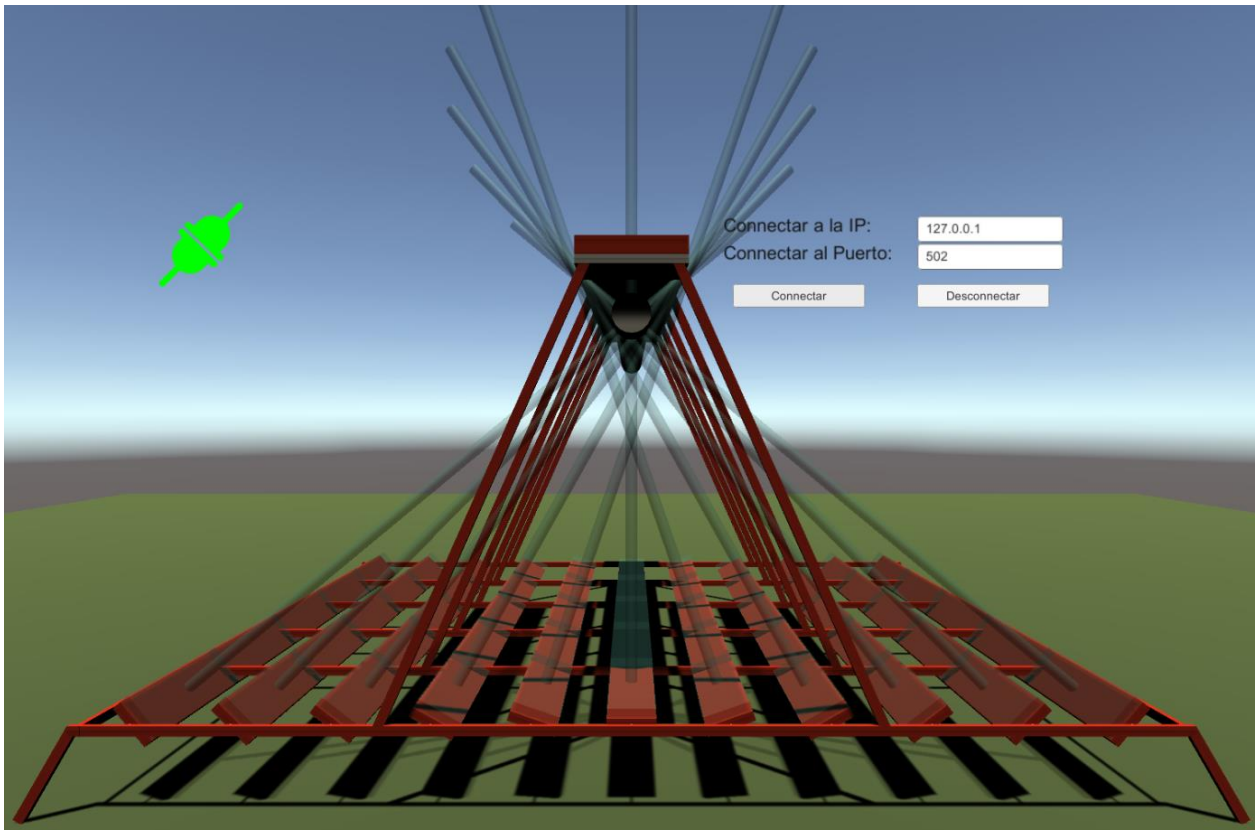

Figura 5-2.- Planta calibrada con la primera versión.

Con la segunda versión no había problemas con el sentido de giro inicial, sin embargo, se puso el recorrido más corto para cada espejo, ya que el movimiento de los motores es muy lento y por tanto la simulación tardaría mucho tiempo. En este caso los resultados obtenidos son muy buenos, se puede ver que todos los rayos de los reflejos intersecan en el mismo punto, que es el punto central del sensor.

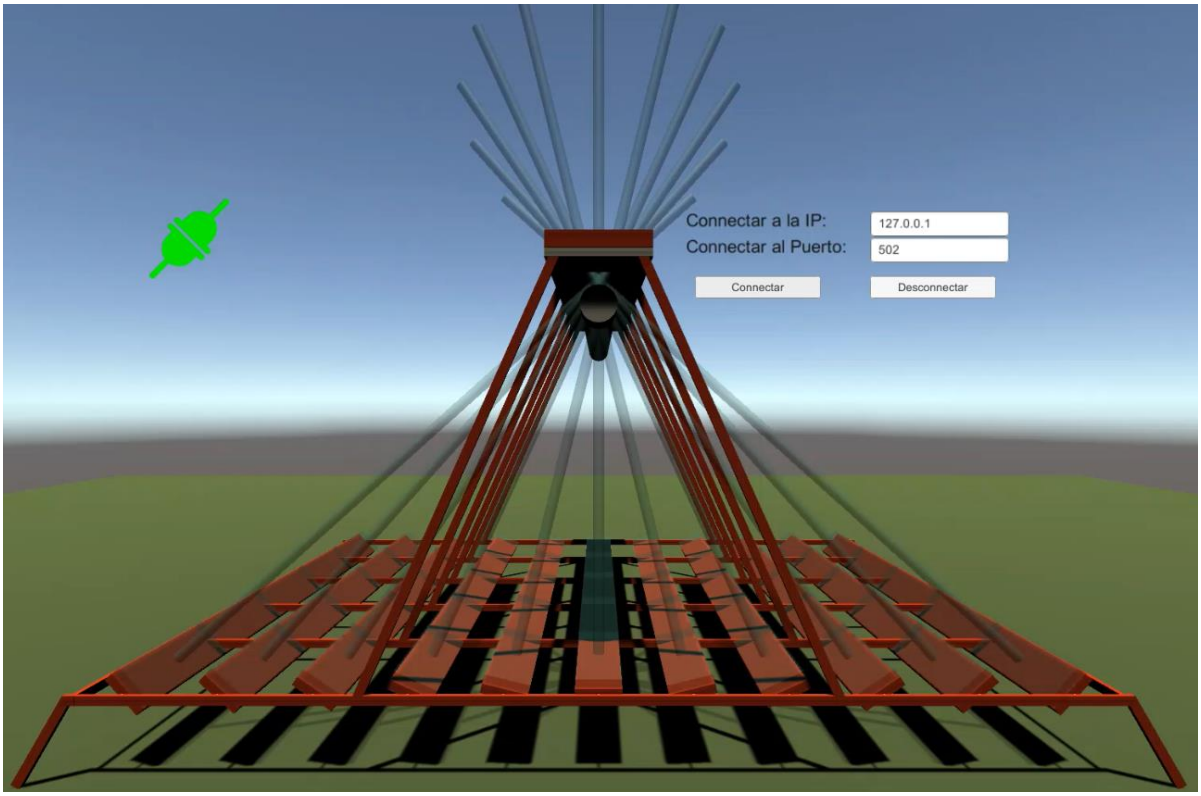

Figura 5-3.- Planta calibrada con la segunda versión.

Visualmente se puede comprobar que los resultados son diferentes y que los obtenidos en el segundo caso han sido mucho mejores, no obstante, también era interesante comparar las diferencias de los ángulos obtenidos.

5.2 Datos calibración ambas versiones

En la calibración con la primera versión los resultados obtenidos fueron:

	Campo 1		
	Posición	Sentido de giro	Salida del sensor
Motor 1	-50	0	1
Motor 2	-45	0	1
Motor 3	-38	0	1
Motor 4	-29	0	1
Motor 5	-19	0	1
Motor 6	0	0	1
Motor 7	19	0	1
Motor 8	29	0	1
Motor 9	38	0	1
Motor 10	45	0	1
Motor 11	50	0	1

Figura 5-4.- Datos calibración primera versión.

Es importante recordar que en la columna central un 0 significa que los sentidos de giro de los motores son nulos y por lo tanto estarían parados.

También veremos los datos obtenidos en la calibración con la segunda versión sin mejorar, en este caso los

resultados fueron, en bastantes casos mejores que los de la primera versión, sin embargo, un fallo como el de la fila 6 no se puede permitir, ya que en la planta virtual esta fila de espejos se encuentra justo debajo del sensor.

	Campo 1		
	Posición	Sentido de giro	Salida del sensor
Motor 1	-45	0	83
Motor 2	-38	0	92
Motor 3	-28	0	64
Motor 4	-18	0	87
Motor 5	-7	0	123
Motor 6	6	0	70
Motor 7	7	0	123
Motor 8	18	0	87
Motor 9	28	0	64
Motor 10	38	0	92
Motor 11	45	0	83

Figura 5-5.- Datos calibración segunda versión sin mejorar.

Con la mejora en el proceso de calibración, el fin era corregir el fallo de esa fila de espejos y mejorar los resultados de las demás, como se vio previamente de forma visual.

	Campo 1		
	Posición	Sentido de giro	Salida del sensor
Motor 1	-48	0	70
Motor 2	-41	0	70
Motor 3	-34	0	70
Motor 4	-24	0	70
Motor 5	-13	0	70
Motor 6	0	0	70
Motor 7	13	0	70
Motor 8	24	0	70
Motor 9	34	0	70
Motor 10	41	0	70
Motor 11	48	0	70

Figura 5-6.- Datos calibración segunda versión mejorada.

Con estos datos se encuentran diferencia de hasta 12 grados, según la prueba realizada, sin embargo, tomando como referencia los resultados obtenidos en la última prueba, la máxima diferencia sería de 6 grados con cualquiera de las otras dos pruebas, esto es un error considerable, aunque no crítico.

El objetivo inicial de proyecto que era un nuevo, funcional y mejorado sistema de calibración, sería el obtenido en la última versión. Además de ello, se cuenta con un simulador de una planta solar virtual de tipo fresnel con comunicación ModBusTCP/IP preparada para Unity Pro XL, y que podrá ser utilizado para cualquier prueba necesaria. Esto resultará muy útil para no tener que parar el funcionamiento de la planta real cuando se quiera probar alguna mejora en el funcionamiento.

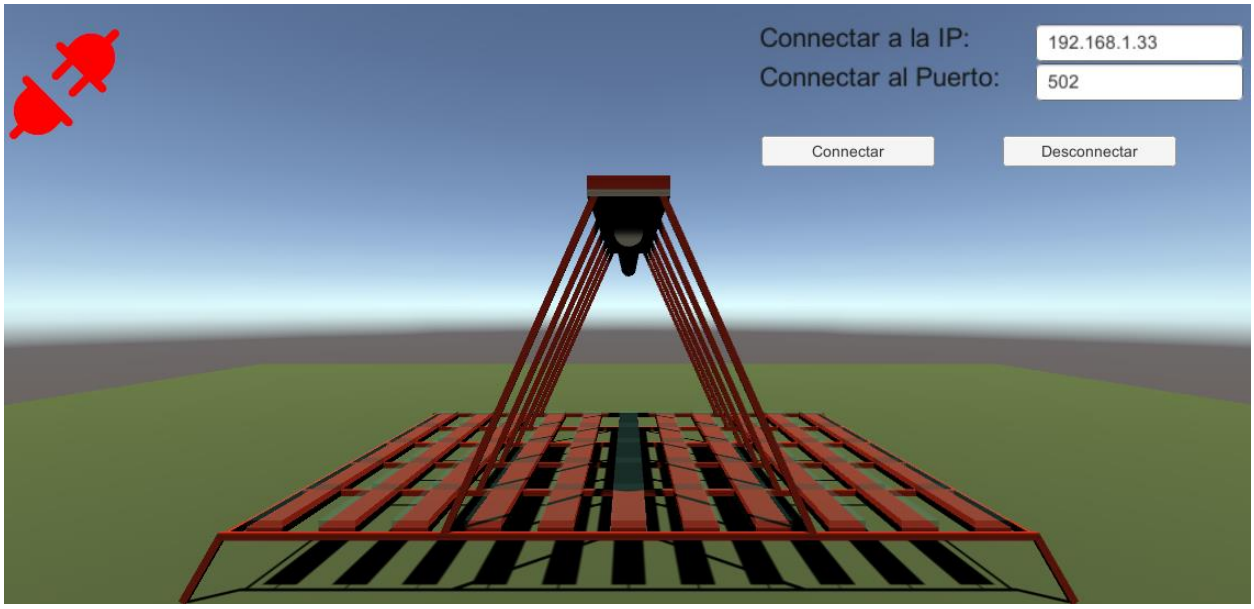

Figura 5-7.- Simulador planta solar tipo fresnel en Unity 3D.

6 CONCLUSIÓN Y TRABAJOS FUTUROS

Este proyecto ha constado de dos partes, un sistema de calibración para una planta solar tipo fresnel programado en Unity Pro XL y la creación de un simulador de una planta solar de este tipo en Unity 3D, además de una comunicación ModBus sobre TCP/IP para usar ambas partes conjuntamente.

Conjunto con el sistema de calibración, también se ha limpiado el código de variables sin utilizar, se han utilizado nombres explicativos para estas y se han documentado las entradas y salidas del autómata, para un mejor entendimiento de su funcionamiento. También se comprobó que servomotores habían dejado de funcionar y se encontró el fallo en la comunicación CANopen entre el campo y el autómata.

El desarrollo de un simulador funcional ha servido para entender mejor el funcionamiento completo de la planta y de distintos tipos de comunicación que se utilizan en la industria. También se ha aprendido un lenguaje nuevo como es C# y el uso de un entorno de desarrollo de videojuegos.

Al utilizar un motor gráfico para el desarrollo de videojuegos, este proyecto puede ser exportable en una aplicación. Esto y algunas modificaciones permitirían el uso de esta planta sin la necesidad de contar con el software de Unity 3D ni con el proyecto original. Realizando esto se podría tener un simulador de una planta solar de tipo fresnel muy fácil de instalar y de utilizar.

Se podría utilizar para temas docentes, ya sean prácticas de una asignatura o un proyecto para los alumnos, estos aprenderían mucho más pudiendo aplicar a un sistema virtual lo que están estudiando. Además, tal y como está programado el simulador actual, hay muchísimas posibilidades para la parametrización del sistema, por lo que se podría experimentar como afectarían diversos cambios en la planta solar. Por ejemplo, en este proyecto se ha podido comprobar cómo ha afectado el cambio en el funcionamiento del sensor.

Haciendo uso de un desarrollador de videojuegos tan potente como es Unity 3D, la capacidad de mejorar el proyecto casi no tiene límites, solo dependería de la capacidad de programar en el lenguaje que se utiliza y el tiempo necesario. Algunas podrían ser:

- Mejoras estéticas con el uso de softwares de modelado 3D.
- Aumento de la velocidad de la comunicación ModBus sobre TCP/IP.
- Introducción de un calendario solar para parametrizar correctamente el reflejo de los espejos según la posición del sol en cada instante del día.
- Añadir funcionamiento de la generación de calor en el captador y estudiar diversas técnicas de control.
- Añadir el resto de la instalación real, desde el flujo del agua por las tuberías hasta la máquina de absorción.

REFERENCIAS

[1] «eltiempo,» [En línea]. Available: https://www.eltiempo.es/videos/sabias-que/en-donde-hay-mas-horas-de-luz-en-espana.

[2] «Energias Renovables,» [En línea]. Available: https://www.energias-renovables.com/termosolar/la-torre-mas-alta-del-mundo-para-20200129.

[3] Reve, «Reve,» 24 Agosto 2012. [En línea]. Available: https://www.evwind.com/2012/08/24/aprueban-la-primera-central-termosolar-de-chile-con-360-mw/.

[4] R. Events, «Reuter Events,» [En línea]. Available: https://newenergyupdate.com/csp-es/el-crecimiento-de-la-energia-solar-en-india-liderado-por-la-fv-pone-la-mirada-en-los.

[5] «Schenider Electric,» [En línea]. Available: https://www.se.com/es/es/.

[6] Á. R. Portillo, Desarrollo de plantas virtuales mediante Unity 3D®. Interconexión Modbus/TCP-IP con PLC industrial, Sevilla: Universidad Loyola, 2018.

[7] H. Hashem, «CSP today,» 3 Septiembre 2012. [En línea]. Available: http://es.csptoday.com/tecnolog%C3%ADa/%C2%BFpuede-superar-la-fresnel-la-cilindroparab%C3%B3lica.

[8] J. M. Torre Murillo, Modelo de parámetros distribuidos en Ecosimpro de un captador solar tipo Fresnel proyecto Fin de Carrera, Sevilla: Universidad de Sevilla, 2012.

[9] E. COMUNICACIÓN, «Eadic,» 10 Enero 2017. [En línea]. Available: https://www.eadic.com/que-es-un-automata-programable/.

[10] U.-E. 50325-4:2002, «CiA 301».

[11] CiA, «CAN in Automotion,» [En línea]. Available: https://www.can-cia.org/.

[12] U. Technologies, «Unity,» [En línea]. Available: https://unity.com/es.

[13] Modbus, «MODBUS APPLICATION PROTOCOL SPECIFICATIONV1.1b3. 2012.,» [En línea]. Available: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

[14] J. G. Galdón, «UModbusTCP,» [En línea]. Available: http://nodlag.com/umodbustcp/.

[15] «Modbus Server Simulator,» SRE-Solutions, [En línea]. Available: http://easymodbustcp.net/modbus-server-simulator.

[16] U. Technologies, «Unity forum,» [En línea]. Available: https://forum.unity.com/.

[17] O. Planas, «Energia solar,» 13 Mayo 2015. [En línea]. Available: https://solar-energia.net/energia-solar-termica/componentes/concentrador-solar.