ECE445 - SENIOR DESIGN

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Twilight - Individual Progress Report

Author:
Naren Sivagnanadasan

Date: November 6, 2017

Contents

1 Introduction

1.1 Project Overview

Twilight is a project looking to develop a self-organizing and reconfigurable lighting system. Current intelligent lighting systems have various drawbacks that prevent them from being significantly more useful that standard lighting systems due to the rigidity of the systems due to centralized control, the less than intuitive user experience and the cost of the systems.
The environment people live and work in has a deep impact on many things including mood, psychological health and productivity [2]. Simple changes like having the correct color temperature at different times of the day or or having the environment handle simple background tasks to reduce cognitive load may help people live happier and healthier lives [3]. So it is worth putting in the resources to develop a system that can optimize environments for people without adding extra things to manage.
The project seeks to accomplish a few main goals. Installation and maintenance of Twilight should be trivial to the point where a student can put a fixture up with no help. Twilight will detect and localize all nodes in the system and automatically reconfigure if connection to one is lost. Twilight blocks will run off standard AC wall power and will daisy chain power to one another to reduce cabling.Twilight also looks to be an accessible platform for people to learn about distributed systems.

1.1.1 Design Components

1.1.1.1 Physical Design

Each block is a magnetically suspended wooden box with the same dimensions as a standard ceiling tile, a square with external width 24.5 and internal width 23. The internal sides are covered with with aluminum tape, improving the internal light reflection. A strip of 140 LEDs are wrapped along the these inside edges. The bottom face of the frame is covered by a plastic diffuser mounted to the inside of the frame. A Raspberry Pi Zero with the custom daughter board is mounted to the inside of the block and controls the blocks on-board communications, LEDs and power systems. Two barrel jacks to provide and distribute power are inlaid into the two adjacent sides. Lastly, an RJ45 jack is also inlaid into the frame for serial communication between block

1.1.1.2 Power Supply

Each Twilight block uses an integrated switching power supply that provides 4A at 5v. Using a switching power supply allows for a much smaller package than a traditional transformer based power supply. This allows the power supply to achieve its weight and size requirements. However, a switching power supply is more complex than a standard transformer based one and would push the amount of work for this project out of the

scope of this class. We therefore opted into using an integrated component, the Mean Well IRM-20-5.

1.1.1.3 Control Board

The control board manages power distribution to all the components in the node, the control of the LED strip and it hosts the networking hardware

1.1.1.4 Networking

The networking component has 3 parts, the connection to other nodes, the PIC which handles the routing of messages and the connection to the Raspberry Pi.

1.1.1.5 Software

When the system is powered on for the first time, it will go through a localization protocol where each block will begin to create a map of the system. Blocks will assign itself and propagate the location assignment to its neighbors signing the message with its EEPROM ID and the number of times the message has been propagated. The block will take assignment from the message with the lowest hops. Once the topology of system is established, the system will enter the default functionality initialized on each block.

User control of the system is done through border routers (blocks that are both connected to the system and some sort of external interface e.g. the local network or even a light switch). This will be the gateway to control the systems high level functionality.

It may be the case that a user would want to take down the system and move it, or add a new block or remove a broken block. The self organizing property of Twilight makes it robust to these possibilities. If a block is removed, the system will continue onwards since there is no dependency between blocks in the system.

1.2 Individual Responsibilities and Role

The break up of work is I am handling the networking hardware, protocol development and software. Rauhul is working the full system integration, developing the Pi Hat we are using to host the LED and Networking components and the PSU carrier board. We both share the responsibility of the physical hardware design, though the majority of that work is already complete.

2 Individual Design Work

2.1 Changes Since Review

Since the review, we have finalized our hardware design. We are going to continue using our wood frame design. We decided on our light diffuser as a plastic film we are stretching across the frame and have a tentative physical layout of the internals of the system.

We decided that instead of the original idea of a full 12 node system being completed by the end of the semester, we are going to focus on a 4 node system to demonstrate the technology. The idea is that we should still be able to demonstrate the key tenants of the project (Self-Organizing, Reconfigurable, and Accessible) with these 4 nodes and scaling up simply requires manufacturing more nodes.

We also decided on using I2C as the interconnect between the PIC and the controller. We added a level converter to the design since the PIC's operating voltage is 5v and the Pi operates at 3.3V.

We moved the LED driver circuit into a more simple driver, relying more on software to handle timings instead of doing it in hardware. Our original design was complex enough that we felt that even if we spent the full semester designing it, the driver as planned may or may not be completed due to how precise and complex the single wire protocol the LED strips we are using use. In fact we felt if we were to implement our original plan that the simplest way to deploy the design would be to put an FPGA on our control board, which we felt was a bit too out of scope considering the other work to be done and the fact we are a two person team.

2.2 Progress

We have finished the final four physical boxes for the system demo as seen in Figure 1. Slight modifications to those boxes will be needed once the boards for the PSU and control arrive to allow for internal wiring. We also finalized the first version of the control board. The schematic is shown in Figure 2 and the PCB is shown in Figures 3 and 4. Finally, we have started work on the software stack for the project including a simulator for Twilight nodes and initial firmware, networking and application code. We also built a device prototype to start testing software.

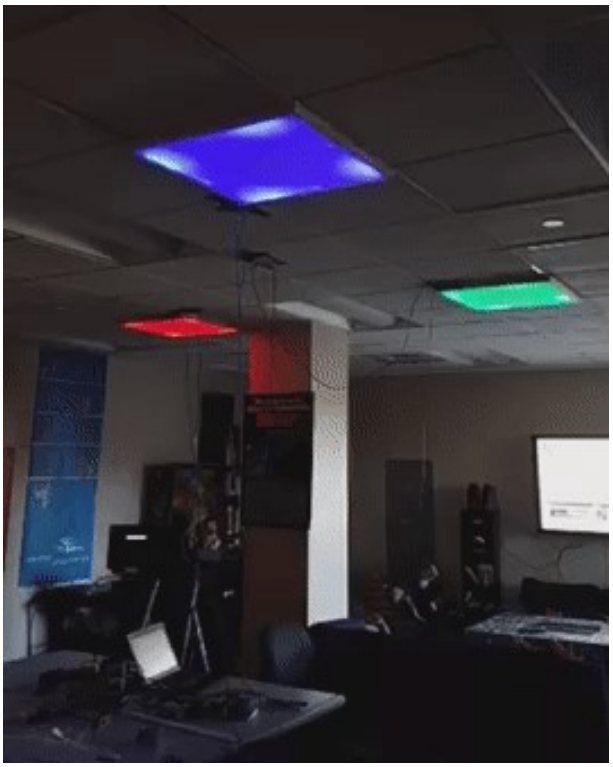

Figure 1: Physical Design of the System

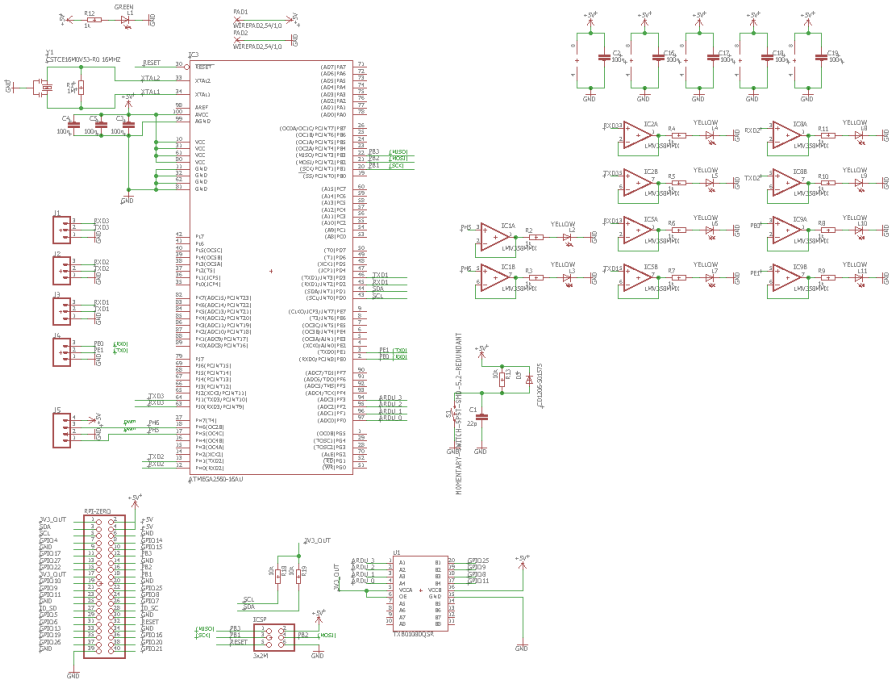

Figure 2: Control Board Schematic

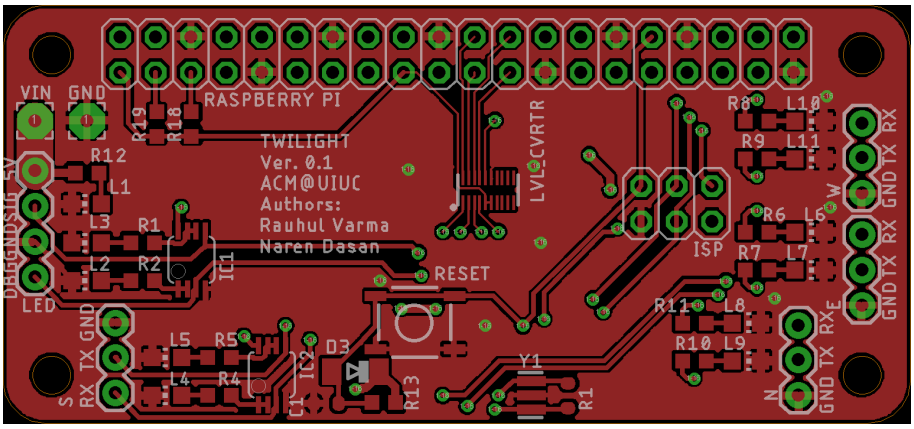

Figure 3: Top layer of the PCB

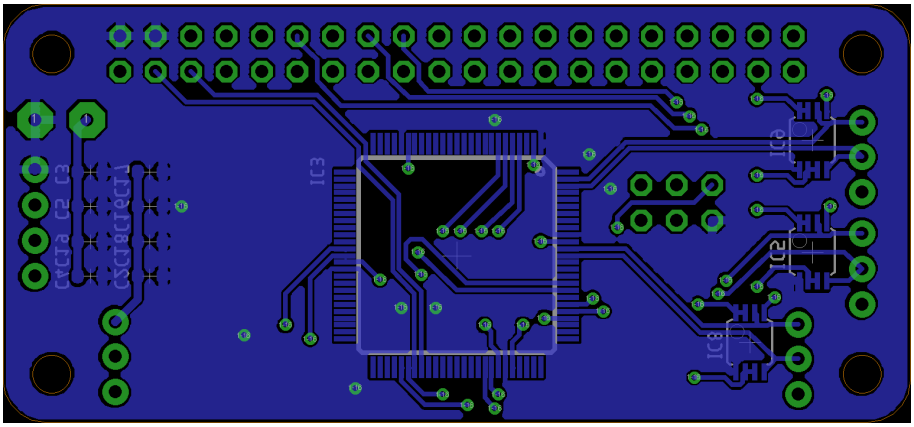

Figure 4: Bottom layer of the PCB

2.3 Testing/Verification

We have two main testing paths, a simulator and a device prototype. The simulator helps test applications at scale (with more nodes than we can afford to prototype). The device prototype allows us to test code that we would actually deploy on devices. The device prototype is only useful until we have functional hardware (which should be next week). However, the simulator could tern into a tool that would remain useful even after the project ends as developers can use it to test code.

2.3.1 Simulation

Because the real control board will not be here for another week, I have begun developing a Twilight Node Simulator so that I can start to write the device control system code without access to the hardware right away. The simulator instead of working over serial connections, works over sockets. There are 4 pairs of sockets connecting to other

simulator processes and a another pair connecting to the prototype device code. An example of some of the code used in the simulator is in Appendix A.

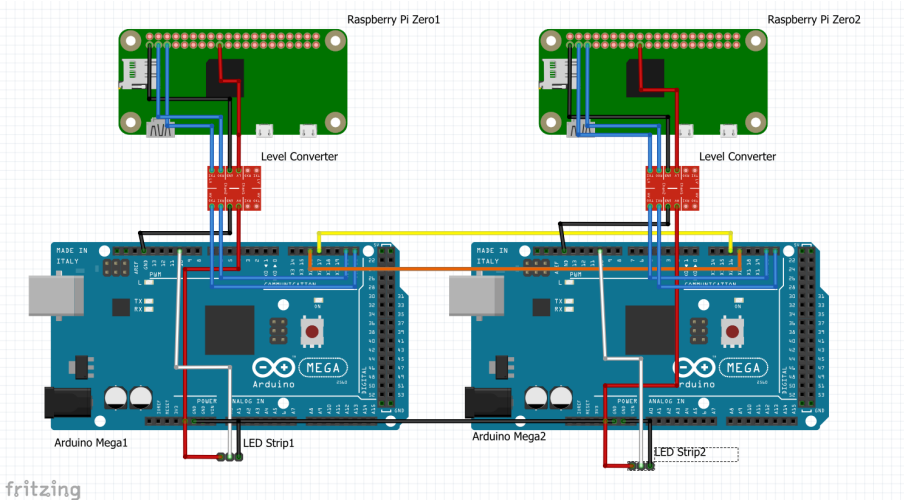

Figure 5: System Prototype

2.3.2 Device Prototyping

We use an Arduino Mega 2560 as a stand in for the control board. The Mega runtime is similar to the runtime we will be using and it has the same 4 serial ports that we will use in the final control board. We also use the ATmega 2560 as our PIC and LED Driver so the Arduino Mega is a convenient prototyping platform. Figure 5 shows the system prototype I built to help prototype the firmware, networking and application code. It shows the Pi connected to the controller board (Arduino) via I2C and a level converter. The two simulated nodes are connected via a serial connection.

2.3.3 Modular Testing

2.3.3.1 Power Supply

We cannot test the power supply until the module arrives, though we have done extensive testing on the power draw of the system (though this has not changed since the design document)

2.3.3.2 LED Driver

We have tested a prototype LED drive through the device prototype above. We can successfully select a color on the device and have the color display it on the strip.

2.3.3.3 Networking

As mentioned above we are able to send messages between two devices via a serial connection.

2.3.3.4 Software

We still have not tested the protocol layer as its not written yet, but in order to verify it is function it must pass the following test: The system must be able to power on and being executing the default program. In doing so the system will establish localization and display a pattern.

2.3.3.5 PCB

The PCBs have not yet arrived though to verify the board we will use the test software we developed for the device prototype. The PCB should function exactly the same as the prototype so the same test program for the device prototype should work for the PCBs.

2.3.4 Contingencies

2.3.4.1 Power Supply

If the PSU does not supply enough current our contingency is to reduce the load by artificially limiting the brightness and concentration of the LED strips. If it is too heavy, we may resort to mounting the PSU separately to the rest of the node.

2.3.4.2 LED Driver

For the LED driver, if for some reason we are unable to control the LEDs with the PCB and its not because of a bug in the software, it is most likely due to a bug in the PCB design, if time permits we may look to get another board revision done. If not we will look to use some of the extra debugging pins we broke out on the PCB to try and recover the board.

2.3.4.3 Networking

If we cannot use the PCB to network between nodes for reason, we could look at using the integrated networking stack on the Pi to do the networking and localization.

2.3.4.4 Software

If we cannot get localization working in software, then we can hard code it, though it means that we have not fulfilled one of the key objectives of the project. If we cannot

get the an abstract application layer working then we can hard code an application for the demonstration.

2.3.4.5 PCB

In the outcome that we cannot get the PCB working in enough time and the PCB is completely nonfunctional, we may resort to using a Teensy as our control board.

2.3.4.6 Results

We are currently able to send a message between the two Pis via the serial connection. We can also control color of the LED strip with the Raspberry Pi via the control board. Now we need to develop more generic code and a runtime so that a developer can create apps without having to worry about the networking.

3 Conclusion

3.1 Self-Assessment

We are currently a week behind our proposed schedule as we are still waiting for our boards to arrive which were sent out last week. Our software is on schedule though the choice to make a simulator means some extra unplanned work. We have shown that the network layer of our system works though our system prototype and now development of the application layer is underway.

As to load, Rauhul has been doing more of the work that is under time constraints since we need the board fast enough to address any mistakes and as most of my work is software development and prototyping which only comes to the forefront once the boards arrive. So, Rauhul for the last couple weeks has been working on the PCB after my initial design. I have been building the prototypes detailed above and been writing iterations of the networking, firmware and the application code.

3.2 Plan for Remaining Work

For the remainder of the semester, I am going to finish the rest of the software and we need to populate the boards and create the final iterations of the lights. We then need to prepare the demonstration for the final presentation.

Week	Tasks
11/7/17	Populate Board Address potential PCB bugs + send out new revision
11/14/17	Populate revised PCB Finish PIC Firmware and Networking Software
11/21/17	Finish Application code
11/28/17	Install array Create front-end + demo app
12/4/17	Demo

Table 1: Semester schedule

3.2.1 Goals

3.2.1.1 Ambitious

- Demonstrate a partition tolerant system
- Demonstrate node synchronization - e.g. can change colors synchronously
- Expand past 4 nodes in the system

3.2.1.2 Realistic

- System self organizes - Is plugged in and beings execution of a distributed program automatically
- User can select another app and the program is propagated
- Apps are self contained and simple

References

[1] C. Rowland, ”Whats different about user experience design for the Internet of Things?”, O’Reilly Media, 2017. [Online]. Available: https://www.oreilly.com/learning/whats-different-about-user-experience-design-for-the-internet-of-things. [Accessed: 03- Oct- 2017] pages

[2] W. van Bommel and G. van den Beld, ”Lighting for work: a review of visual and biological effects”, Lighting Research & Technology, vol. 36, no. 4, pp. 255-266, 2004. pages 3

[3] I. Knez, ”Effects of indoor lighting on mood and cognition”, Journal of Environmental Psychology, vol. 15, no. 1, pp. 39-51, 1995. pages 3

[4] N. Farrow, N. Sivagnanadasan and N. Correll, ”Gesture based distributed user interaction system for a reconfigurable self-organizing smart wall”, Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction - TEI ’14, 2013. pages

[5] H. Ishii, H. Kanagawa, Y. Shimamura, K. Uchiyama, K. Miyagi, F. Obayashi and H. Shimoda, ”Intellectual productivity under task ambient lighting”, Lighting Research and Technology, 2016. pages

[6] R. Kller, S. Ballal, T. Laike, B. Mikellides and G. Tonello, ”The impact of light and colour on psychological mood: a cross-cultural study of indoor work environments”, Ergonomics, vol. 49, no. 14, pp. 1496-1507, 2006. pages

[7] J. Kim, J. Ko and M. Cho, ”A Study of Integrated Evaluation of System Lighting and User Centered Guideline Development - Focused on the Lighting Design Method for Office Space -”, Korean Institute of Interior Design Journal, vol. 23, no. 6, pp. 78-86, 2014. pages

[8] D. Park, Y. Lee, M. Yun, S. Song, I. Rhiu, S. Kwon and Y. An, ”User centered gesture development for smart lighting”, HCI Korea 2016, 2016. pages

[9] F. Tan, ”User-in-the-loop smart lighting control system”, Masters, Delft University of Technology, 2016. pages

[10] D. Burmeister, A. Schrader and B. Altakrouri, ”Reflective Interaction Capabilities by Use of Ambient Manuals for an Ambient Light-Control”, HCI International 2016 Posters’ Extended Abstracts, pp. 409-415, 2016. pages

[11] A. Lucero, J. Mason, A. Wiethoff, B. Meerbeek, H. Pihlajaniemi and D. Aliakseyeu, ”Rethinking our interactions with light”, interactions, vol. 23, no. 6, pp. 54-59, 2016. pages

[12] National Electrical Code. Quincy, MA: National Fire Protection Association, 2007. pages

[13] ”Serial Baud Rates, Bit Timing and Error Tolerance”, 2017. [Online]. Available: http://www.picaxe.com/docs/baudratetolerance.pdf. [Accessed: 20- Oct- 2017]. pages

4 Appendix

Listing 1: PIC Simulator Source Code

```
import zmq
import multiprocessing as mp

URL = "tcp://127.0.0.1:"

class PICSim():
        def init(self,
                 device,
                north_rx,
                south_rx,
                east_rx,
                west_rx,
                north_tx,
                south_tx,
                east_tx,
                west_tx):

                    self.CONNECTIONS["DEVICE"] = device
                    self.CONNECTIONS["NORTH_RX"] = north_rx
                    self.CONNECTIONS["SOUTH_RX"] = south_rx
                    self.CONNECTIONS["EAST_RX"] = east_rx
                    self.CONNECTIONS["WEST_RX"] = west_rx

                    self.CONNECTIONS["NORTH_TX"] = north_tx
                    self.CONNECTIONS["SOUTH_TX"] = south_tx
                    self.CONNECTIONS["EAST_TX"] = east_tx
                    self.CONNECTIONS["WEST_TX"] = west_tx

                    self.context = zmq.Context()

                    self.start_server()
                    self.bind_tx()

                    self.inbox = []
                    self.outbox = []

        def startup(self):
                    self.connect_rx()
```

```
            self.TX_proc = mp.Process(target=self.TX)
            self.RX_proc = mp.Process(target=self.RX)
            self.send_proc = mp.Process(target=self.send)
            self.receive_proc = mp.Process(target=self.receive)

            self.TX_proc.start()
            self.RX_proc.start()
            self.send_proc.start()
            self.receive_proc.start()

            self.TX_proc.join()
            self.RX_proc.join()
            self.send_proc.join()
            self.receive_proc.join()

    def start_server(self):


            ''' FIGURE THIS OUT '''

            self.device = self.context.socket(zmq.REP)
            self.device.bind(URL + self.CONNECTIONS["DEVICE"])
            return

    def bind_tx(self):
            self.north_tx = self.context
                                    .socket(zmq.PUSH)
            self.north_tx.bind(URL +
                            self.CONNECTIONS["NORTH_TX"])

            self.south_tx = self.context
                                    .socket(zmq.PUSH)
            self.south_tx.bind(URL +
                            self.CONNECTIONS["SOUTH_TX"])

            self.east_tx = self.context
                                    .socket(zmq.PUSH)
            self.east_tx.bind(URL +
                            self.CONNECTIONS["EAST_TX"])

            self.west_tx = self.context
```

```
                                .socket(zmq.PUSH)
        self.west_tx.bind(URL +
                        self.CONNECTIONS["WEST_TX"])

        return

    def connect_rx(self):
        self.north_rx = self.context
                                .socket(zmq.PULL)
        self.north_rx.connect(URL +
                        self.CONNECTIONS["NORTH_RX"])

        self.south_rx = self.context
                                .socket(zmq.PULL)
        self.south_rx.connect(URL +
                        self.CONNECTIONS["SOUTH_RX"])

        self.east_rx = self.context
                                .socket(zmq.PULL)
        self.east_rx.connect(URL +
                        self.CONNECTIONS["EAST_RX"])

        self.west_rx = self.context
                                .socket(zmq.PULL)
        self.west_rx.connect(URL +
                        self.CONNECTIONS["WEST_RX"])

    def send_msg(self, dest, msg):
        '''Test RX/TX part'''
        self.outbox.append({dest, msg})
        return

    def recieve_msgs(self):
        '''Test RX/TX part'''
        msgs = self.inbox
        self.inbox = []
        return msgs

    def send(self):

        ''' FIGURE THIS OUT '''
```

```
            while true:


    def receive(self):

            ''' FIGURE THIS OUT '''

    def RX(self):
            while true:
                    msg = self.north_rx.recv()
                    if msg != None:
                            self.inbox.append(msg)
                    msg = self.south_rx.recv()
                    if msg != None:
                            self.inbox.append(msg)
                    msg = self.east_rx.recv()
                    if msg != None:
                            self.inbox.append(msg)
                    msg = self.west_rx.recv()
                    if msg != None:
                            self.inbox.append(msg)

    def TX(self):
            while true:
                    if len(self.outbox) != 0:
                            msg = self.outbox.pop
                            if msg.dest == "NORTH":
                                    self.north_tx
                    .send(msg.content)
                            elif msg.dest == "SOUTH":
                                    self.south_tx
                    .send(msg.content)
                            elif msg.dest == "EAST":
                                    self.east_tx
                    .send(msg.content)
                            elif msg.dest == "WEST":
                                    self.west_tx
                    .send(msg.content)
                            else:
                                    exit(1)
```

The remainder of the code can be found here: https://github.com/acm-uiuc/twilight