



Group 3

Industrial Programming

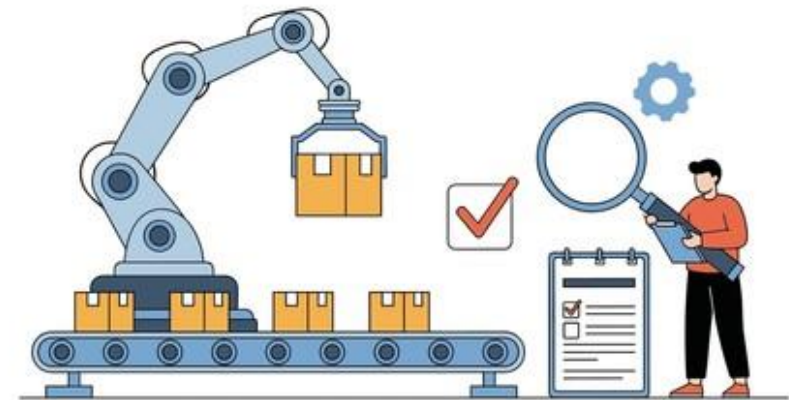
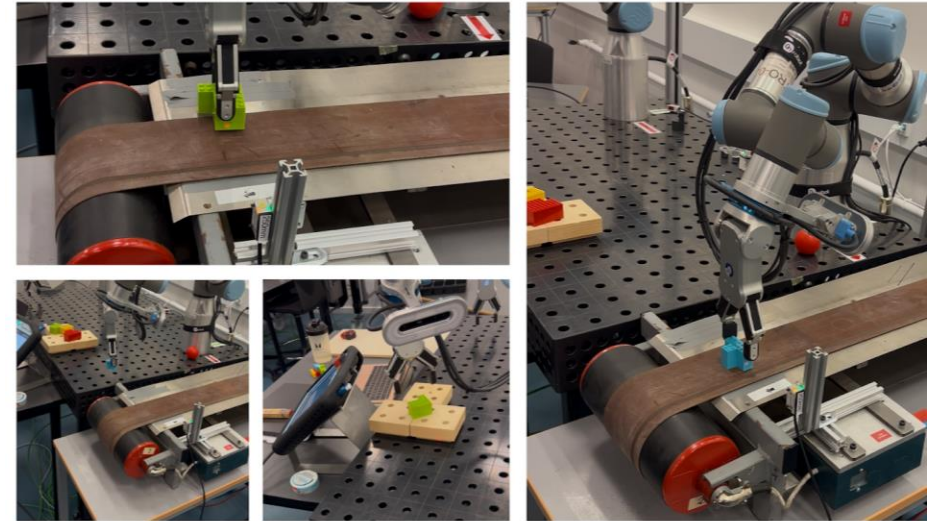
Christian Olesen s235657

Francisco Noppenau s235669

Simon B. Rasmussen s235654

Project case:

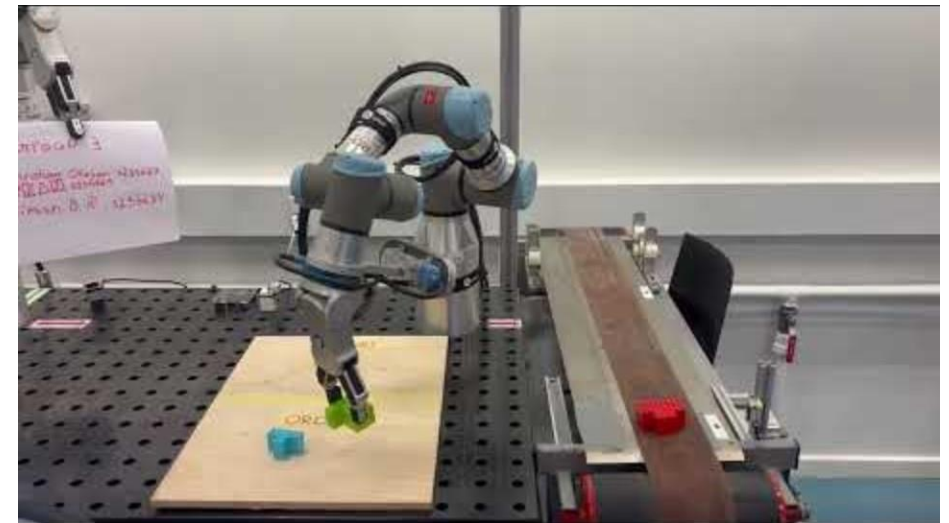
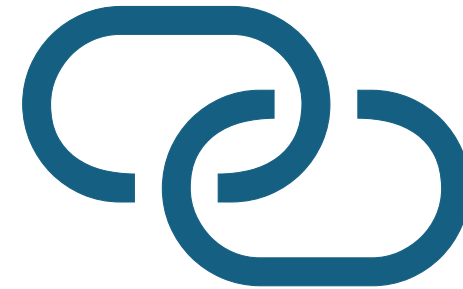
This project solves cases where production lines require automatic, reliable sorting of products for different customers. It replaces manual handling with a camera-guided robotic system that improves efficiency, accuracy, and scalability in industrial workflows.



Video Link

<https://www.youtube.com/watch?v=uZlzbW6h2ql>

[Group 3 - Industrial Programming Robot
sorter](#)



How did we solve it
?

Day 1-3 “Rookie Mistakes”

- Day 1-3 was about figuring out how we wanted to go about the project.

Day 1:

- Initial Camera setup
- Integration of Hardware components

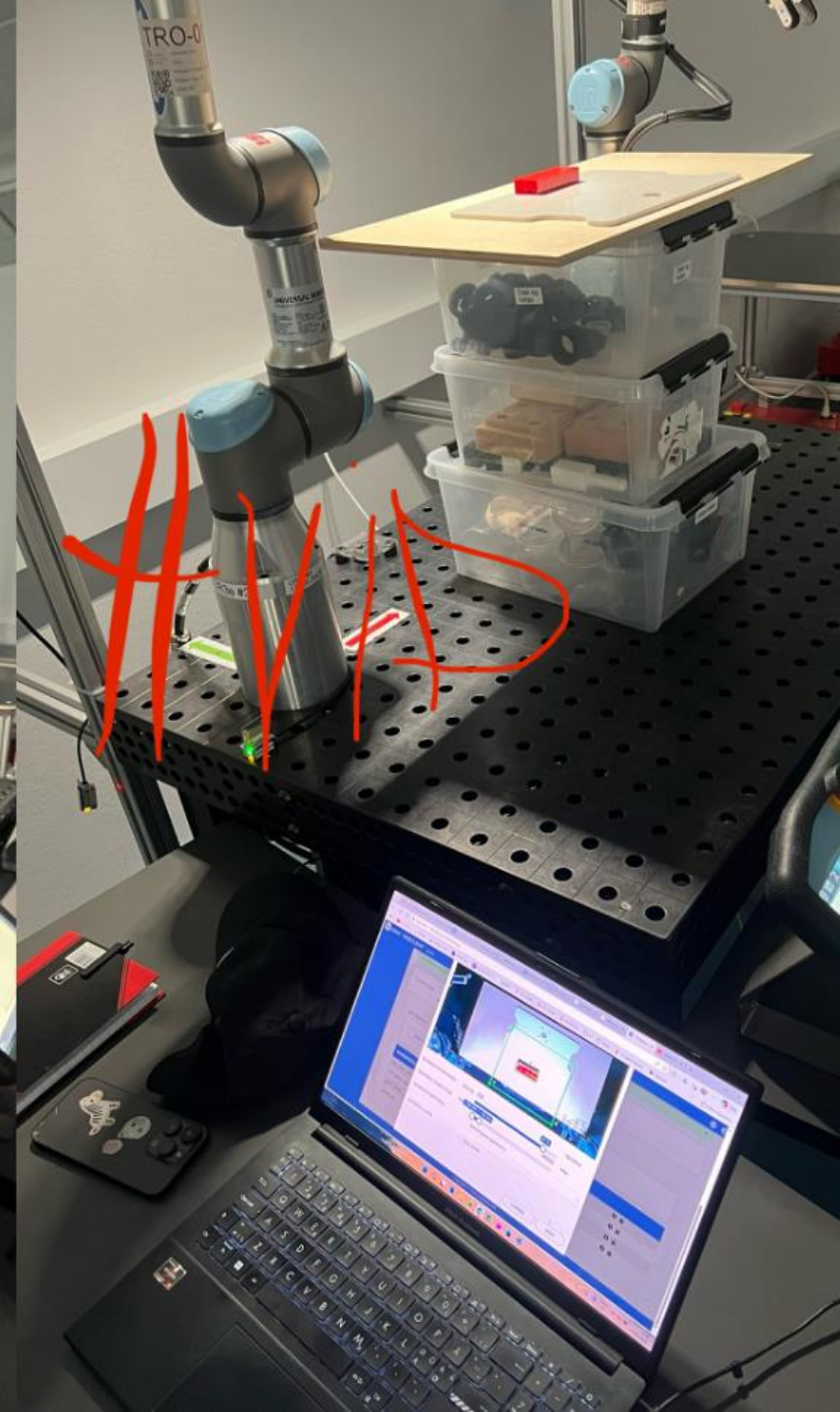
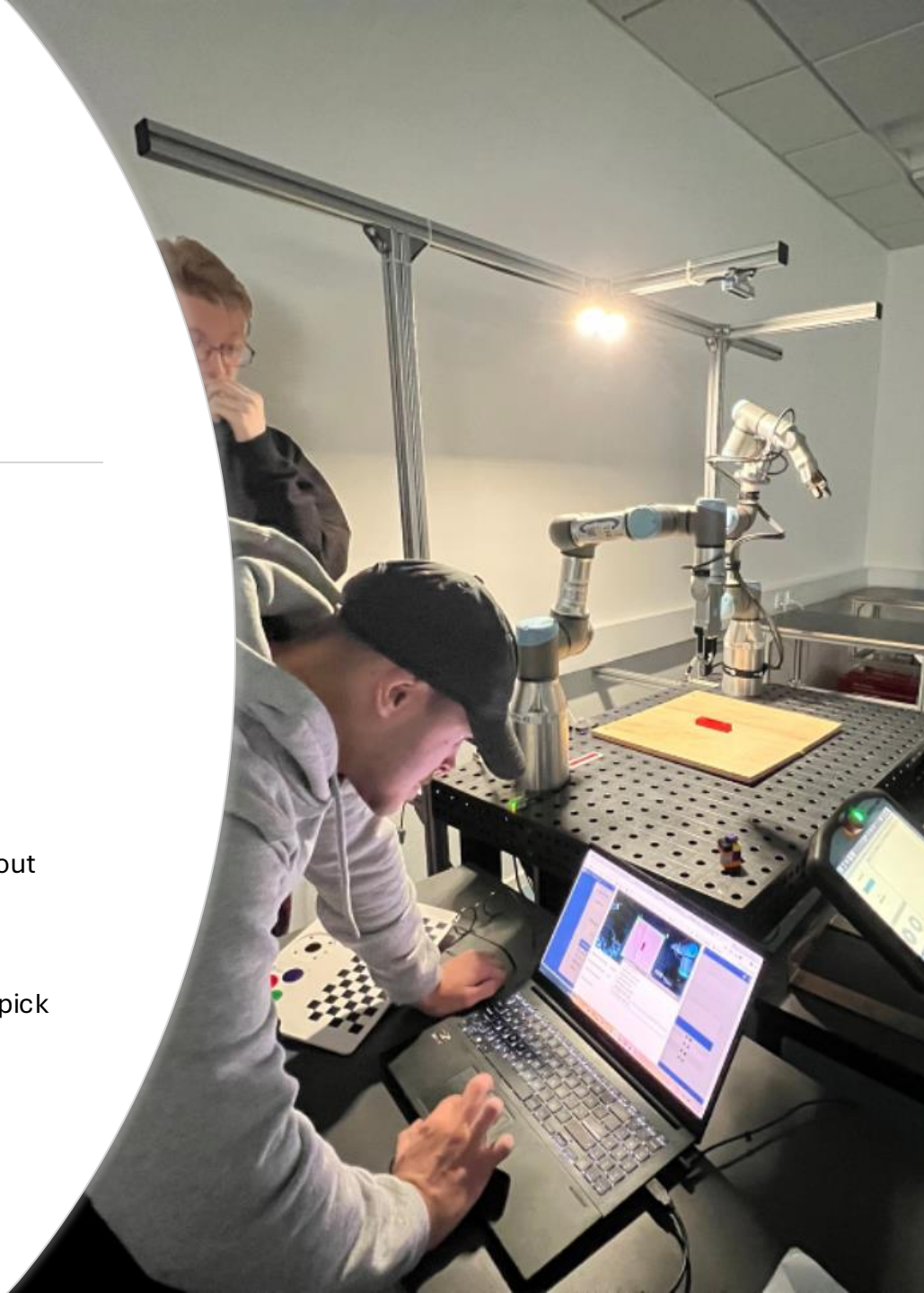
Day 2:

- Camera on arm
- Pendant programming
- Time not wasted; Lessons learned about surroundings and environment

Day 3:

Once we were setup:

- Inverse kinematics error – alternative pick paths
- Consistency is key



Day 4-6 "Progress"

Day 4:

- Trying to make sense of exported file from pendant
- Fewer waypoints

Day 5:

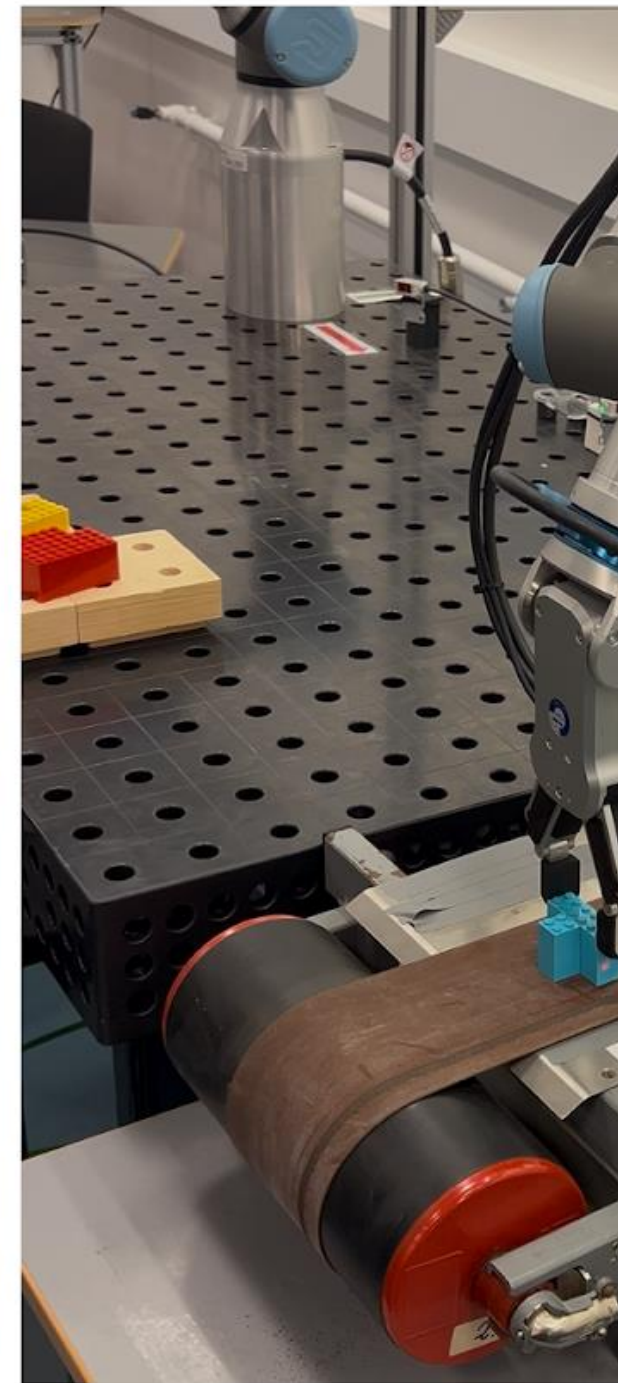
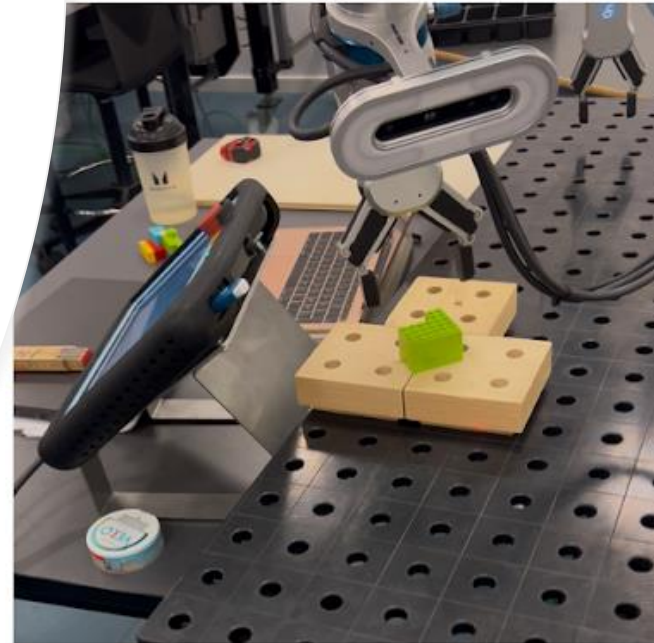
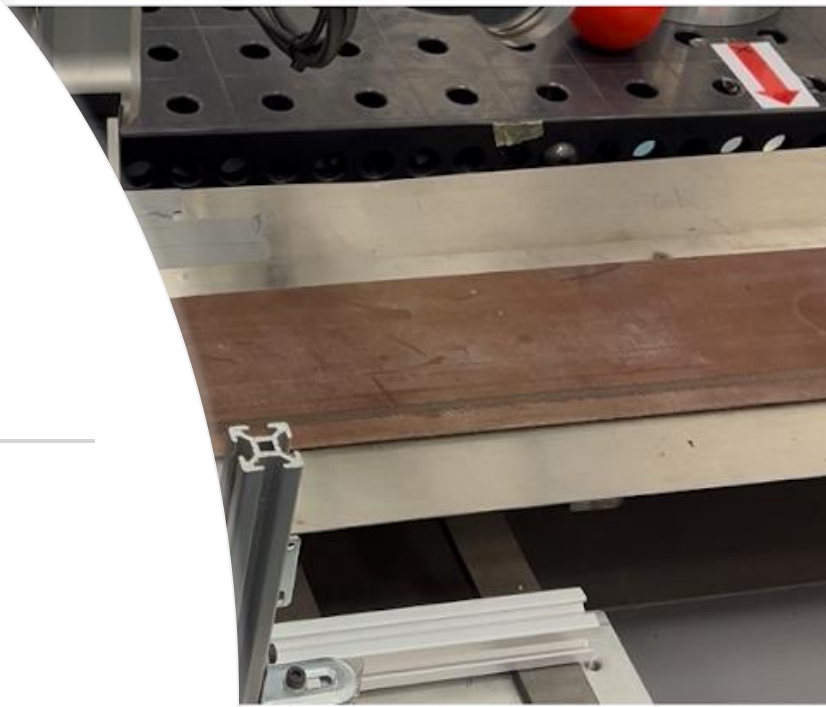
- Choosing to be pragmatic
 - A lot of redundant code left

Day 5:

- Extension of Database
 - Customer profiles
 - Orders
- GUI Customization

Day 6:

- Eyes Locate: 15 seconds
- Multiple quantities of the same brick
- Automatic Log out



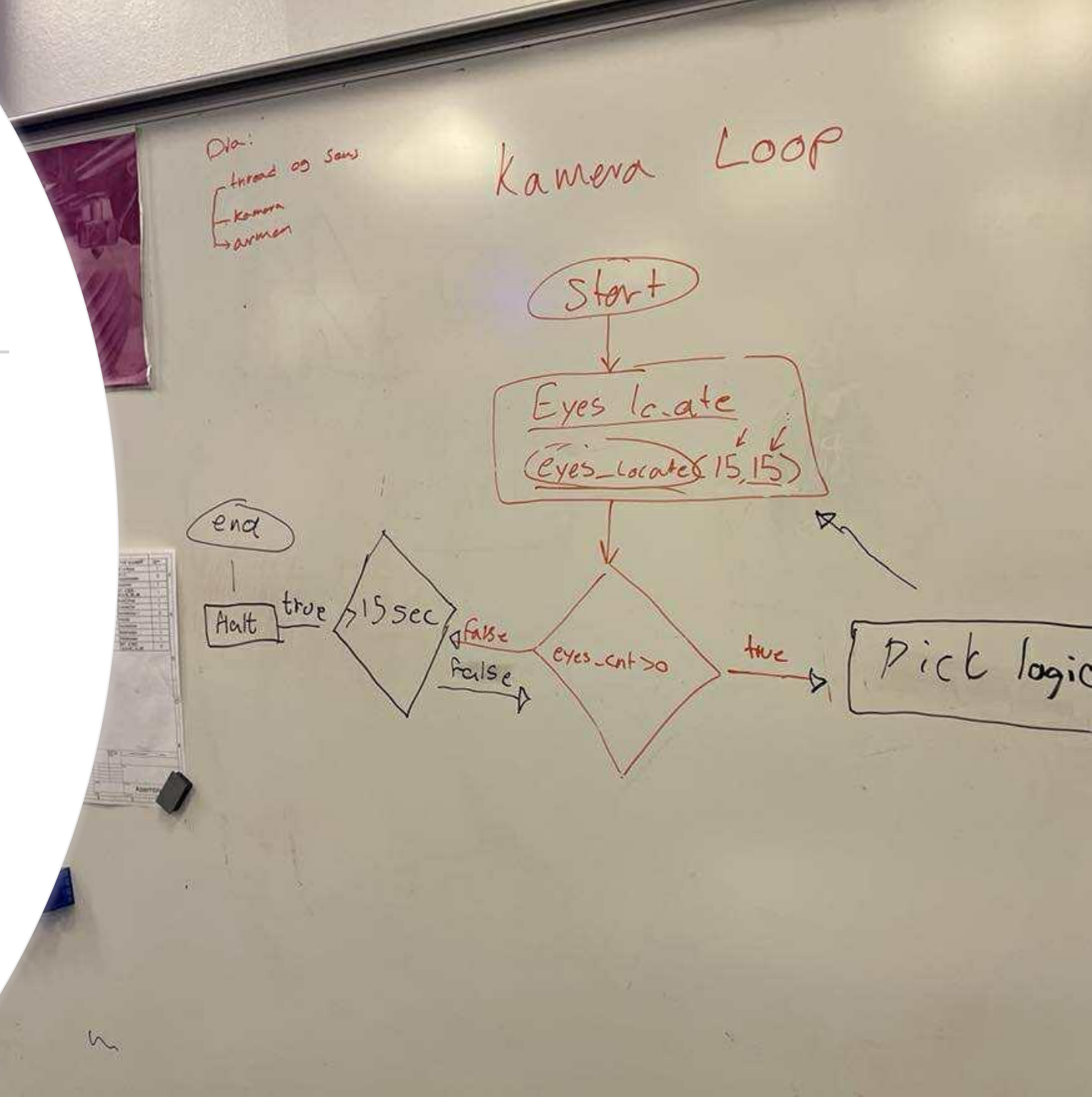
Day 7-8 "KISS"

Day 7:

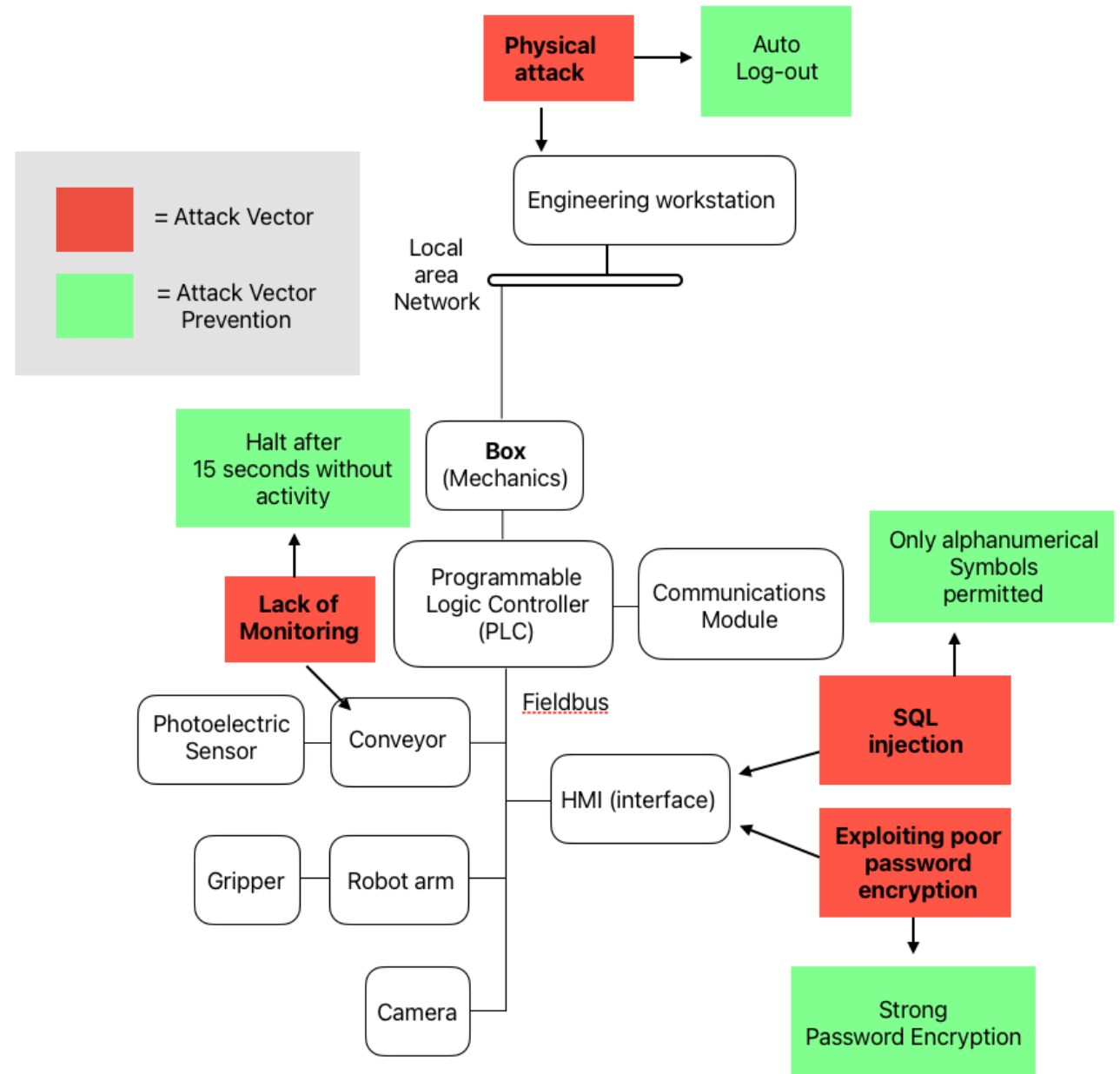
- Details (KISS); halt
 - No need for restart button
 - UI cleanup for better UX
 - Removed old .cs test files

Day 8:

- Flow charts and video – Final product



Program Diagram



Attack Vector and Prevention #1

```
1730     $ 6 "Eyes Locate"
1731     eyes_cnt = 0
1732     eyes_cnt = eyes_locate(15, 15)
1733     if (eyes_cnt > 0):
1734         eyes_workp_cnt = eyes_cnt
1735         # existing pick logic continues
1736     else:
1737         set_standard_digital_out(7, False) # stop conveyor
1738         halt
1739     end
```

**Lack of
Monitoring**



Halt after
15 seconds without
activity

Attack Vector and Prevention #2

Timer

```
_inactivityLogoutTimer = new InactivityLogoutTimer(TimeSpan.FromMinutes(5), HandleInactivityTimeout);
```

Call on resets C-operation

```
4 references  
private void ResetInactivityTimer()  
{  
    if (_currentUser == null)  
        return;  
    _inactivityLogoutTimer.Reset();  
}
```

**Physical
attack**



Auto
Log-out

Attack Vector and Prevention #3

IsLetterOrDigit == true if alphanumeric string

4 references

```
public static bool IsCredentialValid(string? value) =>  
    !string.IsNullOrEmpty(value) && value.All(char.IsLetterOrDigit);
```

**SQL
injection**



Only alphanumeric
Symbols
permitted

Attack Vector and Prevention #4

```
8 public class Authentication(AppDbContext db, PasswordHasher hasher)
9 {
10     public async Task CreateUserAsync(string username, string password, bool isAdmin = false)
11     {
12         var (salt :byte[], saltedPasswordHash :byte[]) = hasher.Hash(password);
13         db.Users.Add(new User
14         {
15             Username = username,
16             Salt = salt,
17             SaltedPasswordHash = saltedPasswordHash,
18             IsAdmin = isAdmin
19         });
20         await db.SaveChangesAsync();
21     }
22 }
```

Exploiting poor
password
encryption



Strong
Password Encryption

Attack Vector and Prevention #5?

Forcing password to be beyond 8 alphanumeric characters.

```
public static bool IsPasswordValid(string? value) =>  
    IsCredentialValid(value) && value!.Length > 8;
```

Total possible passwords

4-character alphanumeric

$$62^4 = 14,776,336 \quad (\approx 1.5 \times 10^7)$$

8-character alphanumeric

$$62^8 = 218,340,105,584,896 \quad (\approx 2.18 \times 10^{14})$$

How much harder is 8 characters than 4?

$$\frac{62^8}{62^4} = 62^4 = 14,776,336$$

➡ An 8-character password is ~14.8 million times harder to brute-force than a 4-character one.

That's the key takeaway.

ERD: Entity Relation Diagram

One-to-Many:

User => Customers

Customers => Orders

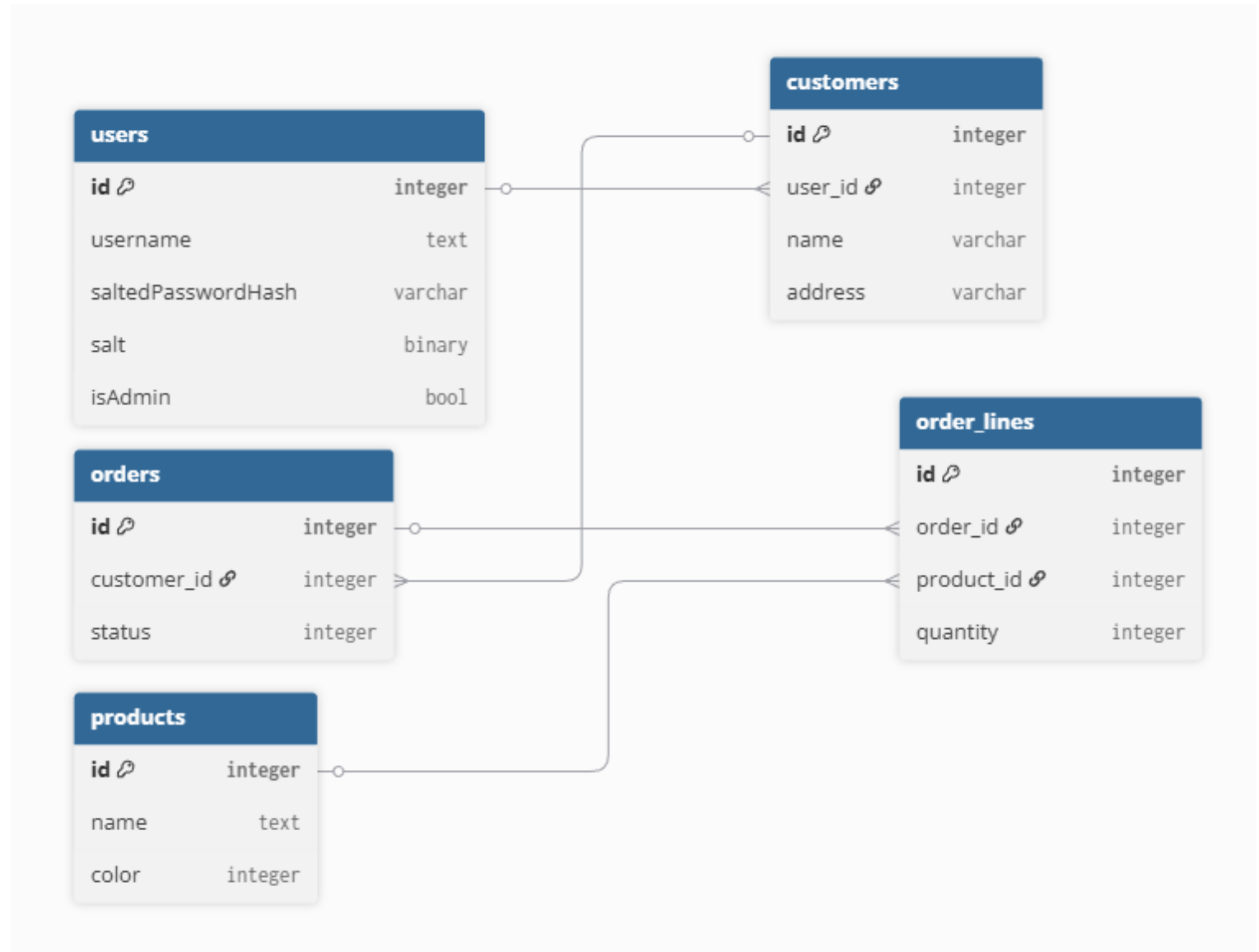
Orders => Order_lines

Product => Order_lines

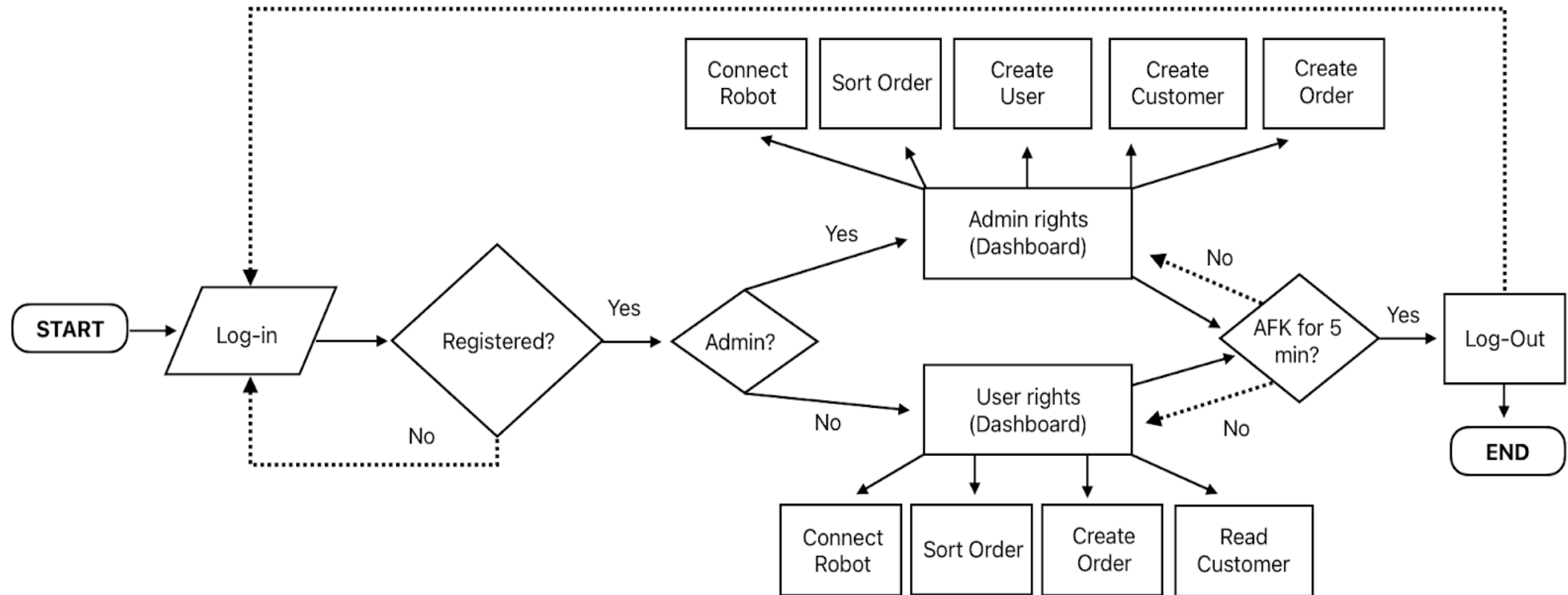
Many-to-Many:

Orders => products by order_lines

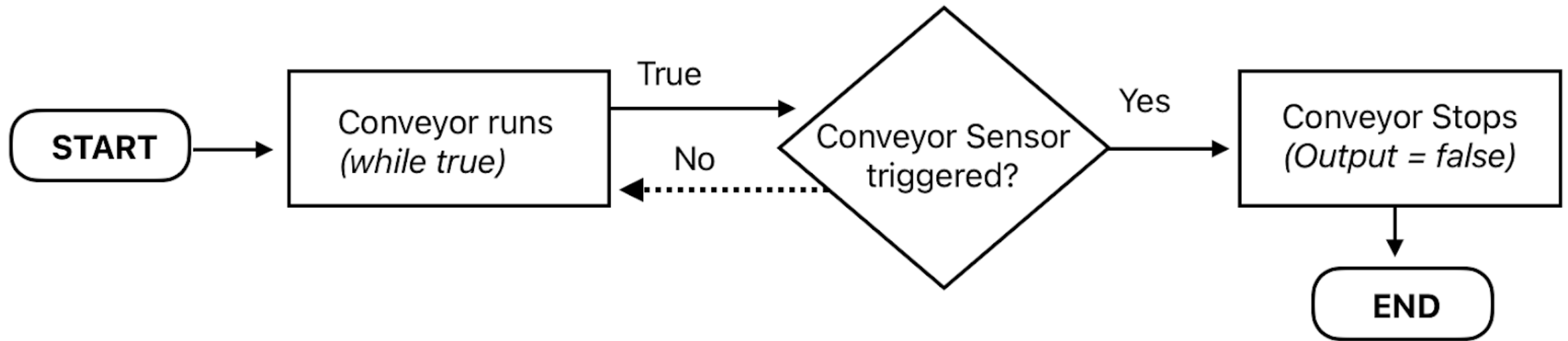
Qty makes it complex



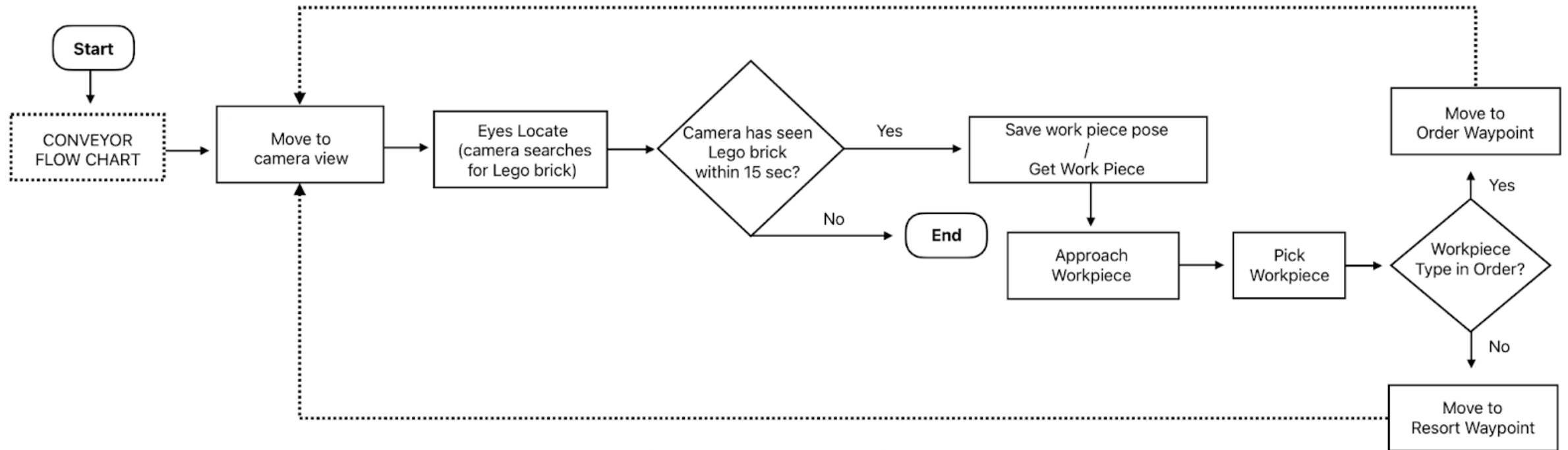
GUI Interaction Flow Chart



Conveyor Flow Chart



Camera and Robot Arm Flow Chart



Robot Tab Flow Chart

