

Exploring The Viability of DIY Mobile Robots, and Technical and Economic Perspective

Y.S. Dighe^{1,*}, Abhijit Gawali², Ajinkya Jadhav³, Gaurav Khairé, ⁵Nisha Pavane⁴

Abstract

This article is intended for on ramping of novices in mobile robotics and shine a new light on feasibility of home-brew robots. Readers are expected to use this resource as a take-off point from which they will get to know where to look and to what extent, along with some of our own recommendations. The topics covered include but not limited to, hardware selection, firmware programming, feedback control, system design, simulation, algorithm selection, software. We also recommend different paths based on readers' goals. In conclusion, as mobile robotics is a vast and complex field, if possible, we recommend beginners to purchase a ready-made robot ideally with ROS support and slowly understand each subsystem by interacting with it in isolation. Else, start with a simple PID controlled differential drive robot then, learn to work with a lidar in simulation to eventually start working in the real world. This study analyzes the balance between cost, performance, and scalability in DIY robotic systems, offering both technical and economic insights. It emphasizes the impact of open-source tools and low-cost hardware in making robotics more accessible and fostering innovation. Furthermore, it provides actionable guidance and suggested learning paths to support beginners in progressing from basic understanding to practical, real-world applications.

Keywords: ROS (Robot operating system), system design, differential drive, PID (proportional integral derivative) control, lidar (light detection and ranging) sensor

INTRODUCTION

Arduino and ESP have ignited a new age of homemade electronics and IoT-based automation. The community of DIY drones has also started to go places, as has that of DIY manipulators, although mobile robotics remains elusive. The Cost of Equipment remains unattainable for many households, and the synthesis of knowledge is not yet available [1]. This study attempts to bridge this gap.

TECHNICAL BASICS

Mobile robotics is an extremely broad and multidimensional field that is overwhelming for novices. It consists of various concepts from the software and hardware fields, such as control theory, path planning, communication architecture, and power management. Therefore, let us cover some absolute basics that are a must to know to navigate the space of mobile robotics [2, 3].

*Author for Correspondence

Y.S. Dighe

E-mail: digheyogeshmk@sanjivani.org.in

¹Assistant Professor, Department of Mechatronics Engineering, Sanjivani College of Engineering, India

^{2,3,4,5}Student, Department of Mechatronics Engineering, Sanjivani College of Engineering, India

Received Date: January 13, 2026

Accepted Date: April 07, 2026

Published Date: June 20, 2026

Citation: Y.S. Dighe, Abhijit Gawali, Ajinkya Jadhav, Gaurav Khairé, Nisha Pavane. Exploring The Viability of DIY Mobile Robots, and Technical and Economic Perspective. Trends in Mechanical Engineering & Technology. 2026; 16(2): 34–41p.

Grid Traversal Techniques

1. *Depth First Traversal:* Keep going down a branch until you cannot go further; then, consider other options.
2. *Breadth First Traversal:* Visit all neighbors first, place them in a visited queue, and visit their neighbors that are not in the visited queue until reaching the target, while keeping a log of cell transitions. Once the target is reached, the logged data are used to backtrack and find the shortest path. (Figure 1)

3. Other Notable Algorithms:

- *Dijkstra*: Breadth-first traversal but with weights for different edges of graphs
- *A**: Dijkstra but, utilized a heuristic function for choice.

Communication Mechanisms

1. *Topics* : Topics are like FM radio; subscribers can choose their publishers. Topics are usually used by sensors and actuators because they need to be non-blocking and have a simpler interface. Logging and visualization are also other use cases (Figure 2).
2. *Services*: Services are just the Client-Server Model, and they carry all the advantages and disadvantages of those here as well. This is good when some processing is needed before fulfilling the request (Figure 3).
3. *Actions*: Actions are preferred for discrete behaviors and tasks that take longer to start. Actions are Heavily used in the Nav2 Package (Figure 4).

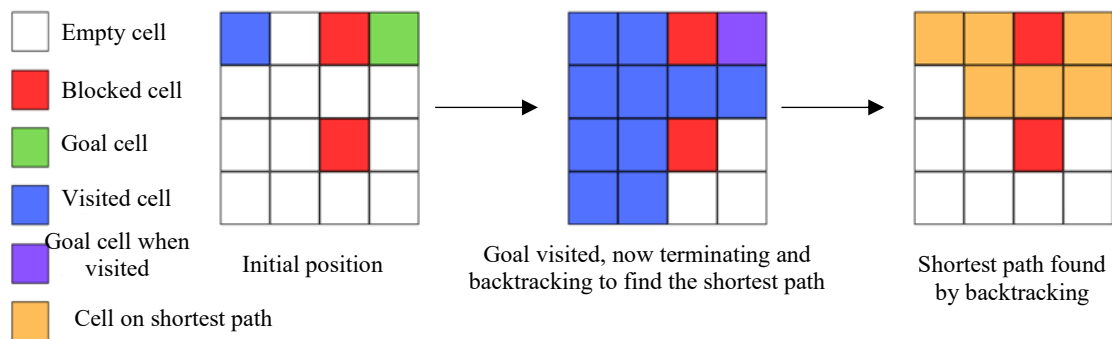


Figure 1. Visualization of shortest path finding using breadth first search.

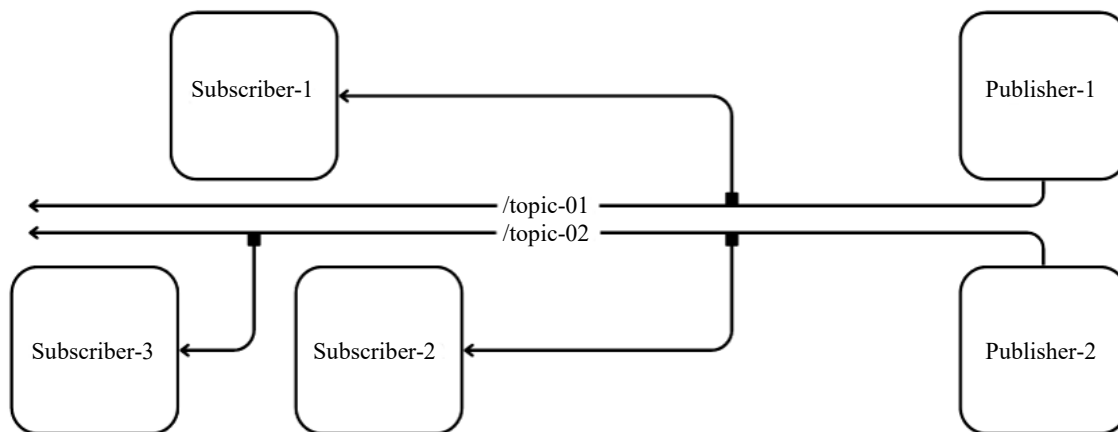


Figure 2. Topics.

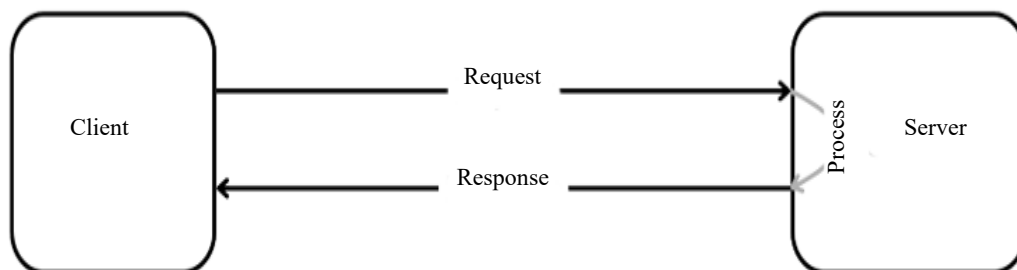


Figure 3. Services.

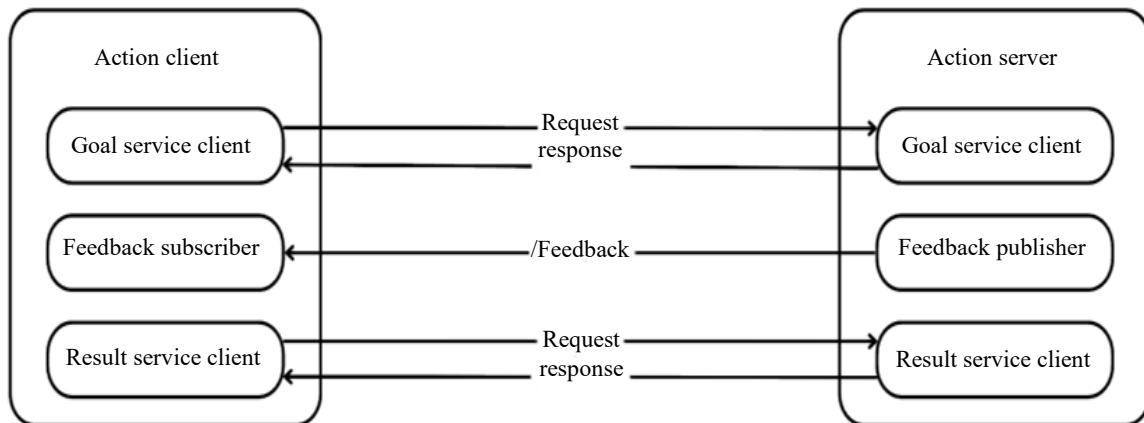


Figure 4. Actions.

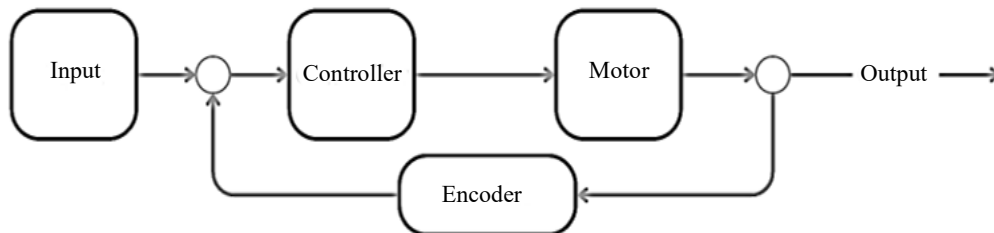


Figure 5. Block diagram of closed loop system.

Control System & Algorithm

A system is a collection of elements/components connected to achieve a certain output from a certain input. When such a system is used to control parameters such as position, speed, and flow, it is called a control system (Figure 5).

Types of Control Systems:

- *Open loop:* These systems do not have feedback; they are simple, economical, and prone to mistakes in the availability of environmental noise. These are usually not robust enough for navigation or robotics but are effective in simple tasks where interference can be predicted and compensated for beforehand.
- *Closed Loop:* These systems have feedback and are relatively sophisticated compared to their open-loop counterparts. These are heavily used in robotics because of their robustness towards noise, which is abundant in real-world scenarios.

Feedback Control Strategies:

- *Proportional-Integral-Derivative (PID):* These systems provide appropriate error corrections according to a simple expression that is dependent on only the current error, previous error, and sum of errors. Do not confuse this simplicity with a lack of application. It is the most commonly used method in robotics and is sufficient for almost all cases. (Figure6)
 K_p - Response Time & may introduce overshoot
 K_i - Opposes Steady State error & may introduce integral windup
 K_d - Opposes overshooting (acts like friction to control the variable as it approaches the target)

$$u(t) = K_p e(t) + \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$
- *Linear Quadratic Regulators (LQR):* Here, the system dynamics are in the form of linear differential equations, and the cost function is quadratic.
- *Model Predictive Control (MPC):* An optimal control strategy that uses current estimates to produce an output that reduces the overall cost-based model of the plant.

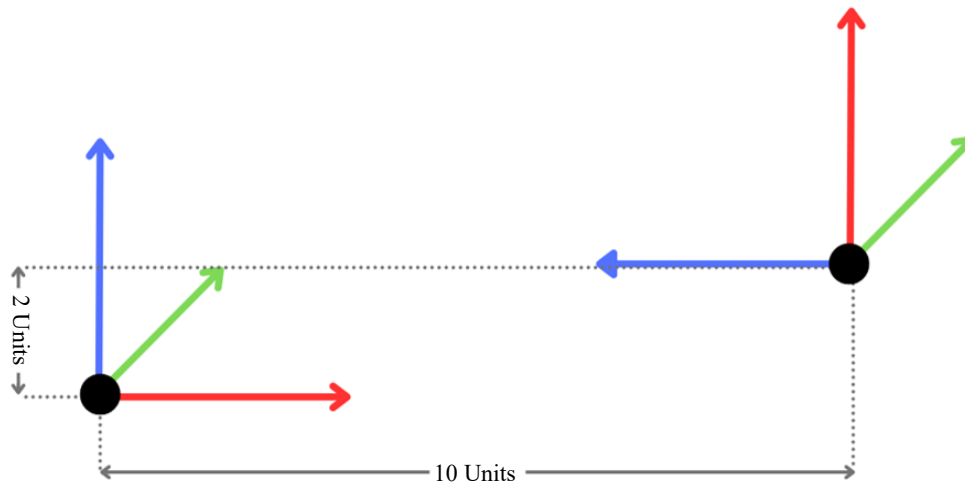


Figure 8. An example of transformation.

Sensor Fusion

The idea behind sensor fusion is to combine two imperfect measurements to reach an estimate that is better than any single measurement. Many types of algorithms/filters are used for this process, and some important ones for robotics are as follows [4].

1. *Complementary Filter*: A simple combination of both outputs in different percentages. For example, in drones, accelerometers and gyroscopes are used for better roll and pitch. In mobile robotics, gyroscopes, magnetometers, and sometimes even encoders are combined for a better estimation of yaw. Accelerometers and encoders can also be combined to estimate odometry. (Figure 7).

$$\theta = a\theta_{acc}\alpha + \theta_{gyr}\cdot(1 - \alpha)a$$

2. *Kalman Filters*: These are sophisticated than simple complementary filters, but the idea is the same, that is, combining multiple flawed measurements for better estimates. There are also different types of Kalman filters, such as the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF).

Transformations

Coordinate transformations are mappings between two links in a multidimensional space. Transformations are expressed in the relative position of the child node with respect to the parent node, usually in 3-Translations and 3-Rotations. There are two main types of transformations.

1. *Static/Fixed*: The transformation is fixed forever, that is, no relative movement is allowed between links.
2. *Dynamic*: These allow for relative movement in one dimension, but multi-degree-of-freedom transforms are also allowed. (Figure 8)

*The following figure shows a transformation of 10 units in the X-axis, 2 units in the Z-axis, and a rotation of -90 ° in the Y-axis, in that order.

SYSTEM DESIGN & HARDWARE

System-Design

Low-level control is offloaded to the controller instead of the processor, which is the proper way to do it (Table 1). This way, the processor can completely focus on computation and communication alone and isolate the expensive components such as the processor and lidar from the high-voltage components such as motors and drivers (Figure 9). The processor communicates with the controller via serial communication (SPI is an alternative) [5].

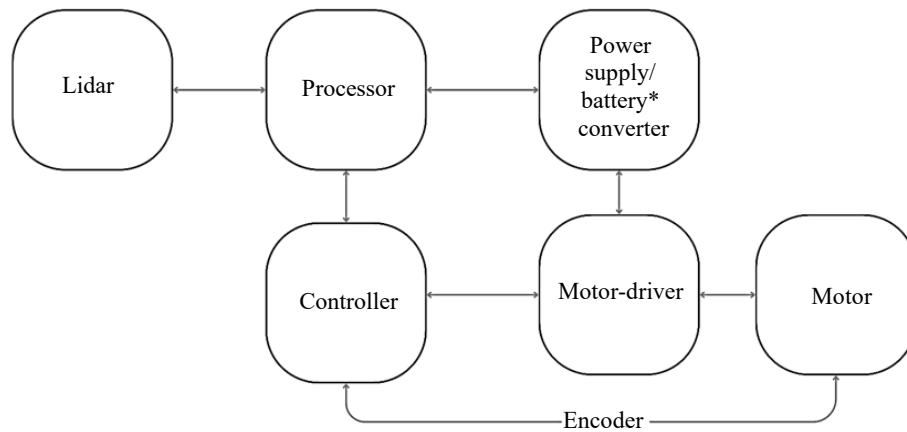


Figure 9. System design.
Hardware

Table 1. Hardware selection.

Component	Recommendation	Reasoning	Alternatives or considerations
1. Microcontroller	Arduino Uno R3 (~300 Rs for clone)	Community support, learning resources & convenience	STM32F103C8T6 ("BluePill") (~200 Rs); ESP32 (~500 Rs)
2. Microprocessor	Raspberry Pi 5 – 4 GB RAM (~6K Rs)	Community support, learning resources & up-to-date	Raspberry Pi Rev. 2 (~5.5K Rs); Raspberry Pi 4 – 4 GB RAM (~5.5K Rs)
3. Motors (Pair)	12V N20 – 52 RPM (~1K Rs)	Torque & Encoders	Any as per chassis and weight
4. Motor Driver	TB6612FNG (~200 Rs)	Less power consumption	L293D (~200 Rs); L298N (~200 Rs)
5. LiDAR	RPLidar A1 (~8.5K Rs)	Price, examples & learning resources	Any as per your budget
6. Battery	Battery: 3S Lithium Polymer (~1–2K Rs)	Capacity and discharge rate	Lithium-ion battery packs (~100 Rs per cell)
7. Buck Converter	XL4015 – 5A (~100 Rs)	Cheap	Any, but make sure it supports adequate current (>5A)
8. Wheels	a. Castor wheel (~50 Rs) b. Driven wheels (Pair) (~300 Rs) c. Couplings for driven wheels (Pair) (~200 Rs)	Necessity	Pick with appropriate sizes and compatibility
9. Chassis	Acrylic (~200 Rs) / Plywood (~50–200 Rs)	Easy to work with consumer tools and yet firm enough	Hard foam

**NOTE: Novices should stick with our recommended controller and processor, since the project itself has plenty of complexity it isn't advised to compound that by choosing core hardware that doesn't have a ton of community support or learning material, alternatives are mentioned for awareness.*

**Few tools that are not part of the robot but are recommended. Drill; Power Supply (can be extracted from an old Computer for free).*

Table 1 summarizes the hardware selection for the proposed robotic system. The selected components include the microcontroller, microprocessor, motors, motor driver, LiDAR sensor, battery, buck converter, wheels, and chassis. The components were selected considering their cost, availability, compatibility, power requirements, and suitability for the system. Arduino Uno R3 and Raspberry Pi 5 are recommended for control and processing, while N20 geared motors and TB6612FNG are used for movement. LiDAR provides distance-sensing capability, and the battery and buck converter ensure appropriate power delivery to the system. The table also lists alternative components that can be considered based on budget, availability, and application requirements.

SOFTWARE

Communication Framework

Robot Operating System

An ecosystem of compatible tools that streamlines robotics development. It provides powerful visualization support in the form of Rviz and is also supported by most off-the-shelf robots and simulators. Python and C++ are the supported languages [6]. Although it now supports Windows, we highly recommend that beginners stick with Linux because of the currently available learning material.

* Some Important Concepts:

- *Universal Robot Description Format (URDF)*: Used to define a robot using an XML file
- *XML Macros (XACRO)*: Utilized to reduce repetition in URDF files
- *Communication Mechanisms*: Topics, Services and Actions (As introduced in Section II.B)

*Some Important Packages of Note:

- *Robot-State-Publisher*: for static transforms
- *Joint-State-Publisher*: for dynamic transforms
- *Teleop-Twist-Keyboard*: for teleoperation

Alternatives for Communication

These can be used, but novices are still advised to stick with ROS.

- *ZeroMQ*
- *Mobile Robot Programming Toolkit (MRPT)*

Simulator

1. *Gazebo*: An open-source physics simulator that is the most widely used by the ROS community. Its file format is SDF(Simulation Description Format), and URDF can be easily translated to this. There are two versions; the classic version is end-of-life, but most learning resources still use it and should be given some consideration. However, for future compatibility, the current version is recommended [7].
2. *Alternatives*:
 - *Webots*: Cross-platform, even support web
 - *Nvidia-Isaac*: Better compatibility with Nvidia Components
 - *AWS-RoboMaker*: For Cloud features
 - *Mujoco*: Cross-platform & simple

ROS2-CONTROL

ROS2-Control is a real-time kernel for the interaction of ROS2 with real-world hardware. It abstracts away the need to write drivers for each component.

* There are three types of hardware:

- *System*: reads and writes and collections of joints
- *Sensor*: Only reads, and single joint
- *Actuator*: Only writes and single joint

* Hardware Component Lifecycle

- *Unconfigured (on_init, on_cleanup)*: Hardware is Initialized, Communication isn't
- *Inactive (on_configure, on_deactivate)*: Communication is Initialized
- *Active (on_activate)*: Hardware has Electricity flowing through it
- *Finalized (on_shutdown)*: Interface is ready for unloading

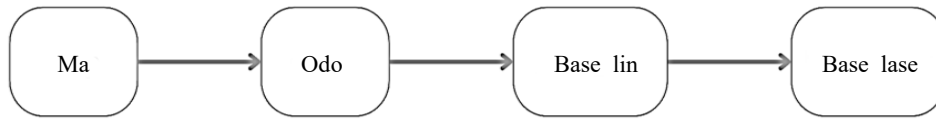


Figure 10. Transform tree for nav-2.

* *Mock Components*: Trivial simulations of hardware components. This is very useful for a quick sanity check and greatly improves the iteration time [8].

* *Firmware*: Some are available on the web; we recommend *ros_arduino_bridge* by Josh Newans although it is highly recommended to try writing your own. To write your own, simply write a UART interface with a PID controller for a differentially driven system.

NAVIGATION-2

Navigation-2 is a ROS-based navigation stack that makes it easy to initiate mobile robot navigation. The actions mentioned in Section II. B are heavily used in the Nav-2 Package (Figure 10).

Transforms of Note for First-Time Robot Setup

- *base_link*: Rotational Center of the body
- *odom*: Fixed frame relative to starting position of robot
- *map*: World Fixed frame for globally-consistent representations of distance

Planner Server

The planner Server is responsible for computing the robot path. The following are some important plugins and algorithms supported by Nav-2.

- *NavFn planner*: Dijkstra or A*
- *Smac 2D planner*: 2D A* using four or eight neighborhoods with smoother and multi-resolution queries
- *Theta Star Planner*: Uses the line of sight to create non-discreetly oriented path segments

Slam Toolbox

Slam Toolbox is a package that provides a set of tools and capabilities for 2D SLAM (*Simultaneous Localization and Mapping*) which is highly compatible and commonly utilized with Nav-2

CONCLUSIONS

DIY mobile robotics has come a long way in terms of user-friendliness, but it is still not near enough to the likes of DIY IoT, Drones, or even CNC. Platform developers and maintainers are also currently more focused on implementing the latest cutting-edge features than on user comfort, as they themselves are working in research organizations and corporations with those demands. However, the future looks promising as the hardware could be purchased at about 20 K–25 K Rs (about the cost of a budget 3d-printer) and the software is becoming more user-friendly [10]. As for learning paths, learners with software and algorithm concerns should start working with simulations first. learners with more hardware concerns should get a ready-made bot if possible and learn about sub-systems in isolation; otherwise, they should start with a simple differential drive PID-controlled robot on a microcontroller and slowly graduate to a microprocessor by adding a LiDAR and using ROS2, ROS2-Control, and Nav-2.

REFERENCES

1. Build a Mobile Robot with ROS. [ONLINE]. Available: <https://articulatedrobotics.xyz/category/build-a-mobile-robot-with-ros>
2. Self Driving and ROS 2 - Learn by Doing! Plan and Navigation. [ONLINE]. Available: <https://www.udemy.com/course/self-driving-and-ros-2-learn-by-doing-plan-navigation>

-
3. ROS 2 Nav2 [Navigation 2 Stack] - with SLAM and Navigation. [ONLINE]. Available: <https://www.udemy.com/course/ros2-nav2-stack/>
 4. ROS-2 Documentation. [ONLINE]. Available: <https://docs.ros.org/en/jazzy/index.html>
 5. Nav-2 Documentation. [ONLINE]. Available: <https://docs.nav2.org/>
 6. ROS-2 Control Documentation. [ONLINE]. Available: <https://control.ros.org/jazzy/index.html>
 7. Choi JY, Ahn S, Kim D, Heo J, Yun WJ, Hong S, Bae S, Ahn SH. Exploring challenges and opportunities in manufacturing and intelligence for future robotics. *International Journal of Precision Engineering and Manufacturing*. 2025 Sep;26(9):2203-22.
 8. Cavallo F, Limosani R, Manzi A, Bonaccorsi M, Esposito R, Di Rocco M, Pecora F, Teti G, Saffiotti A, Dario P. Development of a socially believable multi-robot solution from town to home. *Cognitive Computation*. 2014 Dec;6(4):954-67.
 9. Woern AL, Pearce JM. Distributed manufacturing of flexible products: Technical feasibility and economic viability. *Technologies*. 2017 Oct 30;5(4):71.
 10. Pedersen SM, Fountas S, Have H, Blackmore BS. Agricultural robots—system analysis and economic feasibility. *Precision agriculture*. 2006 Sep;7(4):295-308.