دانشگاه شهید بهشتی
دانشکده مهندسی و علوم کامپیوتر

فرود ربات پرنده بدون سرنشین بر روی هدف متحرک مبتنی بر رویکرد های هوش مصنوعی در محیط شبیه ساز

پروژه کارشناسی مهندسی کامپیوتر

دانشجو:
سید مصطفی اطیابی

استاد راهنما:
دکتر آرمین سلیمی بدر

تابستان ۱۴۰۳

چکیده

این پروژه به توسعه و ارزیابی یک سیستم خودکار برای هدایت ربات‌های پرنده به سمت باندهای فرود از طریق شبیه‌سازی در محیط ویباتز و پردازش تصاویر با استفاده از مدل YOLOv5 small پرداخته است. هدف اصلی این پروژه، طراحی و پیاده‌سازی یک سیستم هوشمند است که قادر باشد باندهای فرود را شناسایی کرده و ربات پرنده را به‌طور خودکار به سمت آن هدایت کند. در روند تحقیق، چالش‌هایی همچون دقت شناسایی باندها و پایداری کنترل حرکت ربات در شرایط واقعی مورد بررسی قرار گرفته است.

مشکل اصلی در این حوزه، عدم دقت کافی در شناسایی باندهای فرود و کنترل دقیق حرکت ربات در محیط‌های پیچیده است. مدل‌های موجود در شناسایی باندهای فرود محدودیت‌هایی در دقت و سرعت پردازش دارند که باعث ایجاد نیاز به یک راه‌حل کارآمدتر می‌شود. این پروژه با بهره‌گیری از مدل YOLOv5 small برای شناسایی باندهای فرود و استفاده از الگوریتم‌های کنترل حرکت، تلاش کرده است تا این مشکلات را برطرف کند.

روش پیشنهادی شامل استفاده از مدل YOLOv5 small برای شناسایی باند فرود و کنترلر هوشمند برای هدایت ربات پرنده است. نتایج حاصل از ارزیابی نشان می‌دهد که مدل در شناسایی باندهای فرود به‌خوبی عمل کرده، اما نیاز به بهبود در سرعت پردازش و دقت کنترلر وجود دارد. این پروژه به‌طور کلی موفق به دستیابی به اهداف اصلی خود شده و به‌عنوان پایه‌ای برای توسعه‌های آینده در زمینه کنترل ربات‌های پرنده و پردازش تصویر عمل می‌کند.

واژگان کلیدی :ربات پرنده، شبیه‌سازی، YOLOv5, شناسایی باند فرود، پردازش تصویر، کنترلر هوشمند

فهرست مطالب

فهرست شکل‌ها

فصل اول: کلیّات

۱-۱مقدمه

در دنیای امروز، تکنولوژی‌های مربوط به ربات‌های پرنده بدون سرنشین[۱] به سرعت در حال پیشرفت هستند و کاربردهای گسترده‌ای در صنایع مختلف از جمله نظامی، کشاورزی، امداد و نجات و حمل و نقل پیدا کرده‌اند. یکی از چالش‌های مهم در زمینه استفاده از ربات‌های پرنده بدون سرنشین، فرود دقیق بر روی یک هدف متحرک است. این مسئله به دلیل پیچیدگی‌های پویای و نیاز به هماهنگی دقیق در زمان واقعی[۲]، یکی از مسائل برجسته در حوزه رباتیک و کنترل خودکار محسوب می‌شود.

اهمیت پرداختن به این موضوع از آنجا ناشی می‌شود که در بسیاری از کاربردها، به ویژه در عملیات‌های نجات و شرایط اضطراری، امکان فرود سریع و دقیق ربات‌های پرنده بدون سرنشین بر روی اهداف متحرک می‌تواند تأثیر قابل‌توجهی بر کارایی و موفقیت عملیات داشته باشد. برای مثال، در عملیات نجات افراد در دریا، ربات‌های پرنده بدون سرنشین باید قادر باشد به سرعت به موقعیت فرد برسد و بدون خطا بر روی یک هدف متحرک (مثلاً یک قایق یا حتی فردی که در حال حرکت است) فرود بیاید. این مسئله نه تنها در زمینه‌های نظامی و امنیتی، بلکه در صنایع دیگری مانند لجستیک و حمل و نقل نیز اهمیت بسیاری دارد.

آمارها نشان می‌دهند که میزان استفاده از ربات‌های پرنده بدون سرنشین در صنایع مختلف به طور قابل‌توجهی در حال افزایش است. بر اساس گزارشی از انجمن بین‌المللی رباتیک، انتظار می‌رود که بازار جهانی ربات‌های پرنده بدون سرنشینتا سال ۲۰۲۵ به ارزش ۴۵ میلیارد دلار برسد. این رشد سریع نشان‌دهنده نیاز فزاینده به تکنولوژی‌های پیشرفته‌تری است که بتوانند عملکردهای پیچیده‌تر و دقیق‌تری را انجام دهند. [۱]

۱-۲ بیان مسئله

در حوزه فناوری‌های پیشرفته، پژوهش‌ها و مطالعات زیادی بر روی ربات‌های پرنده بدون سرنشین انجام شده است. با این حال، همچنان برخی چالش‌های کلیدی در این زمینه به طور کامل حل نشده‌اند و نیاز به تحقیقات و توسعه بیشتری دارند. یکی از این چالش‌ها، که کمتر به آن پرداخته شده است، توانایی این ربات‌ها در انجام فرود دقیق و ایمن در شرایط متغیر و پویا است. در بسیاری از مطالعات پیشین، تمرکز عمدتاً بر روی کنترل مسیر و پایداری پرواز بوده و مسئله فرود دقیق بر روی اهداف متحرک به صورت جامع بررسی نشده است.

---

[۱] Unmanned Aerial Vehicles - UAVs

[۲] Real-Time

این چالش بیشتر کاربران صنعتی و نظامی را تحت تأثیر قرار می‌دهد که نیاز به عملیات‌های دقیق و ایمن دارند. برای مثال، شرکت‌های لجستیک، نیروهای امداد و نجات، و بخش‌های نظامی که از ربات‌های پرنده بدون سرنشین در عملیات‌های حساس استفاده می‌کنند، به طور مستقیم از این مسئله رنج می‌برند.

این مسئله عمدتاً در مراحل پایانی یک ماموریت، یعنی زمانی که ربات‌های پرنده بدون سرنشین باید برای انجام فرود آماده شود، رخ می‌دهد. در این مرحله، نیاز به هماهنگی دقیق بین حسگرها و سیستم‌های کنترلی برای انجام یک فرود ایمن و دقیق بیش از هر زمان دیگری احساس می‌شود. با ظهور و پیشرفت سریع فناوری ربات‌های پرنده بدون سرنشین و ورود آن‌ها به کاربردهای پیچیده‌تر در سال‌های اخیر، این مسئله برجسته‌تر شده است. هرچه نیاز به عملیات‌های دقیق و پیچیده‌تر افزایش یافته، ضعف‌های موجود در سیستم‌های کنترلی کنونی نیز آشکارتر شده‌اند.

این مسئله به دلیل محدودیت‌های موجود در فناوری‌های کنونی رخ داده است. سیستم‌های کنترلی سنتی، که برای فرودهای ساده طراحی شده‌اند، نمی‌توانند به طور موثر با شرایط متغیر و پویا که در محیط‌های عملیاتی واقعی با آن مواجه هستند، کنار بیایند. همچنین، بسیاری از این سیستم‌ها قادر به پردازش و تطبیق سریع داده‌ها در زمان واقعی نیستند.

راه‌حل‌های موجود، که عمدتاً مبتنی بر الگوریتم‌های کنترلی سنتی و حسگرهای محدود هستند، در مواجهه با پیچیدگی‌های محیطی و حرکت غیرقابل پیش‌بینی اهداف متحرک ناتوان بوده‌اند. این سیستم‌ها معمولاً از دقت و قابلیت اطمینان کافی برخوردار نیستند و نمی‌توانند به طور مداوم و قابل اعتماد در شرایط مختلف عمل کنند [۲][۳].

پروژه حاضر بر استفاده از روش‌های پیشرفته هوش مصنوعی و یادگیری ماشین برای حل این چالش تمرکز دارد. در حالی که بسیاری از تحقیقات پیشین به جنبه‌های دیگر عملکرد ربات‌های پرنده بدون سرنشین پرداخته‌اند، این پروژه به طور خاص به بهبود دقت و کارایی در فرود بر روی اهداف متحرک می‌پردازد. استفاده از الگوریتم‌های پیشرفته برای پردازش سریع داده‌ها و تصمیم‌گیری در زمان واقعی، جنبه‌ای است که در کارهای دیگر به طور کامل مورد توجه قرار نگرفته است.

۱-۳ کلیات روش پیشنهادی

در این پروژه، هدف اصلی توسعه یک رویکرد بهینه برای فرود دقیق ربات پرنده بدون سرنشین بر روی یک هدف متحرک با استفاده از تکنیک‌های هوش مصنوعی و بینایی ماشین است. این رویکرد شامل دو مرحله اصلی است که به ترتیب زیر توضیح داده می‌شود:

- **شناسایی و پیگیری هدف با استفاده از مدل پیش‌آموزش‌یافته YOLOv5s[1]**

در مرحله اول، از مدل پیش‌آموزش‌یافته YOLOv5s برای شناسایی و پیگیری هدف استفاده می‌شود. YOLOv5s (نسخه کوچک YOLOv5) یکی از مدل‌های سریع و کارآمد در زمینه شناسایی اشیاء است که به دلیل طراحی بهینه خود برای سرعت و دقت بالا در شرایط مختلف، به‌ویژه برای کاربردهای با منابع محدود، مناسب است. YOLOv5s بر اساس معماری تک‌مرحله‌ای طراحی شده است که با استفاده از شبکه‌های عصبی کانولوشنی[2] قادر است تا تصویر ورودی را به جعبه‌های مرزی[3] و برچسب‌های کلاس تبدیل کند.

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

شکل ۱-۱ مقایسه مدل های مختلف YOLOv5 [4]

مدل YOLOv5s به دلیل اندازه کوچک‌تر و نیاز کمتر به منابع پردازشی، برای پیاده‌سازی بر روی ربات‌های پرنده بدون سرنشین مناسب است. این مدل با توانایی شناسایی اشیاء در زمان واقعی، امکان پردازش سریع و دقیق تصاویر را فراهم می‌آورد. برای این پروژه، YOLOv5s با استفاده از داده‌های آموزشی شامل تصاویر متنوع از اهداف متحرک آموزش دیده است. این آموزش باعث می‌شود که مدل قادر به شناسایی دقیق موقعیت و ابعاد هدف در فواصل دور باشد و اطلاعات لازم برای مراحل بعدی پردازش را به‌طور موثری فراهم آورد.

---

[1] You only look once version 5
[2] convolutional neural network (CNN)
[3] bounding boxes

- **استفاده از روش‌های سریع‌تر بینایی ماشین برای افزایش دقت و سرعت در نزدیکی هدف**

در مرحله دوم، هنگامی که ربات‌های پرنده بدون سرنشین به هدف نزدیک می‌شود، از تکنیک‌های پیشرفته بینایی ماشین برای بهبود دقت و سرعت پردازش استفاده می‌شود. این تکنیک‌ها شامل موارد زیر است:

- ❖ لبه‌یابی[۱]: برای شناسایی لبه‌های مشخص و ساختارهای هندسی هدف، از تکنیک‌های لبه‌یابی مانند الگوریتم Canny یا Sobel استفاده می‌شود. این تکنیک‌ها به ربات‌های پرنده بدون سرنشین کمک می‌کنند تا با دقت بیشتری شکل و مرزهای هدف را شناسایی کند.
- ❖ تشخیص رنگ[۲]: برای تمایز بین اهداف مختلف یا شناسایی ویژگی‌های خاص هدف، از تکنیک‌های تشخیص رنگ استفاده می‌شود. این روش می‌تواند شامل تبدیل تصویر به فضای رنگی HSV و شناسایی رنگ‌های خاص باشد که به تفکیک دقیق‌تر هدف کمک می‌کند.
- ❖ تشخیص شکل[۳]: برای شناسایی اشکال خاص هدف، تکنیک‌های تشخیص شکل مانند Hough Transform یا Contour Detection به کار می‌روند. این روش‌ها به ربات‌های پرنده بدون سرنشین کمک می‌کنند تا با دقت بالاتری اشکال هدف را شناسایی کرده و موقعیت آن را برای فرود دقیق‌تر تعیین کنند.

۱-۴ ساختار پروژه

این پروژه به صورت زیر ساختاربندی شده است:

- فصل اول: مقدمه‌ای کلی درباره پروژه ارائه می‌دهد. این فصل شامل توضیح مسئله، اهمیت موضوع و مرور کلی روش‌های پیشنهادی است.
- فصل دوم: به توضیح مفاهیم پایه و مرور کارهای مرتبط می‌پردازد. این فصل مفاهیم اصلی را بررسی کرده، پژوهش‌های پیشین را تحلیل می‌کند و نقاط قوت و ضعف منابع غیرپژوهشی مشابه را تحلیل می‌نماید.

---

۱ Edge Detection
۲ Color Detection
۳ Shape Detection

- فصل سوم: روش پیشنهادی و نتایج پروژه را توضیح می‌دهد. در این فصل، ساختار و پیاده‌سازی روش پیشنهادی، نحوه ارزیابی آن و تحلیل نتایج به‌دست آمده بررسی می‌شود.

فصل دوم: مفاهیم پایه و کارهای مرتبط

۲-۱ تعاریف و مفاهیم

❖ **اصول بینایی کامپیوتری**

بینایی کامپیوتری به استفاده از الگوریتم‌ها و تکنیک‌ها برای استخراج و تحلیل اطلاعات از تصاویر و ویدئوها اشاره دارد. این شامل شناسایی اشیاء، پیگیری حرکت، و تحلیل ویژگی‌های بصری است. تکنیک‌های رایج شامل فیلترهای تصویری، تبدیل‌های هندسی، و یادگیری ماشین هستند.

❖ **شناسایی برچسب‌ها**

برچسب‌ها یا "تگ‌ها" ابزارهایی برای شناسایی و ردیابی اشیاء در تصاویر هستند. این برچسب‌ها، مانند QR Codes یا April Tags، الگوهای مشخصی دارند که شناسایی آنها را تسهیل می‌کند. این برچسب‌ها معمولاً شامل نقاط مرجع یا الگوهای هندسی ویژه هستند.

❖ **روش‌های هندسی برای تخمین موقعیت**

برای تخمین موقعیت اشیاء در تصاویر، از روش‌های هندسی مانند حل مسأله Perspective-nPoints (PnP) استفاده می‌شود. این روش‌ها موقعیت سه‌بعدی شیء را بر اساس نقاط دو بعدی در تصویر تخمین می‌زنند. همچنین، الگوریتم‌هایی مانند RANSAC برای تأیید هندسی و حذف نقاط دورافتاده به کار می‌روند.

❖ **تبدیل مختصات و فریم‌های مرجع**

تبدیل مختصات برای تبدیل موقعیت‌ها بین سیستم‌های مختلف ضروری است. این تبدیل‌ها معمولاً با استفاده از ماتریس‌های تبدیل و تکنیک‌های هندسی انجام می‌شود تا موقعیت‌های مختلف در فریم‌های مرجع متفاوت تبدیل شوند.

❖ **مدل‌های شبکه‌های عصبی کانولوشنی**

شبکه‌های عصبی کانولوشنی برای تحلیل تصاویر و شناسایی ویژگی‌ها به کار می‌روند. ساختار شبکه‌های عصبی کانولوشنی شامل لایه‌های کانولوشن برای استخراج ویژگی‌ها، لایه‌های فعال‌سازی برای تقویت ویژگی‌های غیرخطی، و لایه‌های تجمع برای کاهش ابعاد است. این شبکه‌ها با استفاده از الگوریتم‌های بهینه‌سازی مانند Adam یا SGD آموزش می‌بینند.

❖ **یادگیری ماشین**[۱]

یادگیری ماشین به الگوریتم‌هایی اطلاق می‌شود که به سیستم‌ها امکان می‌دهند از داده‌ها یاد بگیرند و پیش‌بینی کنند. این الگوریتم‌ها به سه دسته اصلی تقسیم می‌شوند:

- یادگیری نظارت‌شده: شامل الگوریتم‌هایی است که با داده‌های برچسب‌خورده آموزش می‌بینند و برای پیش‌بینی خروجی‌ها استفاده می‌شوند.
- یادگیری بدون نظارت: به شناسایی الگوها و ساختارهای پنهان در داده‌های بدون برچسب می‌پردازد.
- یادگیری تقویتی: از تعامل با محیط و دریافت پاداش‌ها برای بهبود تصمیم‌گیری استفاده می‌کند.

---

[۱] Machine Learning

۲-۲ تحلیل نقاط قوت و ضعف کارهای مرتبط

در تحلیل و ارزیابی پروژه‌های تحقیقاتی و فناوری، بررسی نقاط قوت و ضعف کارهای مرتبط یکی از مراحل کلیدی به شمار می‌آید. این فرآیند به ما کمک می‌کند تا درک بهتری از دستاوردهای موجود در زمینه مورد نظر پیدا کنیم و با شناسایی نقاط ضعف و کاستی‌های آن‌ها، زمینه‌های جدیدی برای بهبود و نوآوری ایجاد کنیم.

در این بخش، تمرکز بر تحلیل نقاط قوت و ضعف کارهای مرتبط با موضوع تحقیق ماست. ابتدا به بررسی موفقیت‌ها و نقاط قوت این کارها خواهیم پرداخت تا مشخص شود که چه عوامل و روش‌هایی به بهبود عملکرد و دستیابی به نتایج مثبت کمک کرده‌اند. سپس، نقاط ضعف و چالش‌هایی که در این تحقیقات وجود داشته‌اند، مورد بررسی قرار خواهند گرفت تا به شناسایی مشکلات و محدودیت‌های موجود کمک شود. این تحلیل جامع به ما امکان می‌دهد تا با درک عمیق‌تر از وضعیت موجود، رویکردها و راه‌حل‌های بهتری برای پروژه‌های آینده پیشنهاد دهیم و به توسعه علمی و عملی این حوزه کمک کنیم.

پژوهش‌های مرتبط با فرود ربات‌های پرنده بدون سرنشین بر روی سطوح متحرک به طور کلی از دو دهه گذشته به سرعت رشد کرده‌اند. این حوزه به دلیل کاربردهای گسترده‌ای که در صنایع مختلف مانند حمل‌ونقل، نظامی، و امداد و نجات دارد، توجه بسیاری از محققان را به خود جلب کرده است. به‌طور کلی، روش‌های مورد استفاده در این حوزه را می‌توان به سه دسته عمده تقسیم کرد: روش‌های مبتنی بر کنترل، روش‌های مبتنی بر پردازش تصویر، و روش‌های مبتنی بر ترکیب حسگرها.

در یکی از پژوهش‌های برجسته‌ای که توسط گروه UZH Robotics and Perception [5] انجام شده است، تمرکز بر توسعه سیستم‌هایی برای فرود دقیق ربات‌های پرنده بر روی سطوح متحرک بوده است. این گروه که در زمینه رباتیک و به‌ویژه ربات‌های پرنده شهرت دارد، موفق به ایجاد سیستمی شده که ربات‌ها بتوانند به‌صورت مؤثری و با دقت بالا، فرود بر روی سطوح متحرک را حتی با سرعت‌های مختلف انجام دهند. این پژوهش در محیط‌های فیزیکی و با استفاده از آزمایش‌های متعدد، کارایی خود را به خوبی نشان داده است.

یکی از نوآوری‌های مهم در این پژوهش، استفاده از الگوریتم‌های پیشرفته بینایی ماشین و تکنیک‌های لوکالیزیشن برای شناسایی و تعقیب سکوی فرود[1] است. برای افزایش دقت و مقاوم‌سازی الگوریتم در برابر داده‌های ناهنجار، محققان از روش RANSAC برای تأیید هندسی استفاده کرده‌اند. این روش به آن‌ها اجازه می‌دهد تا با فرض دانستن اندازه متریک برچسب،

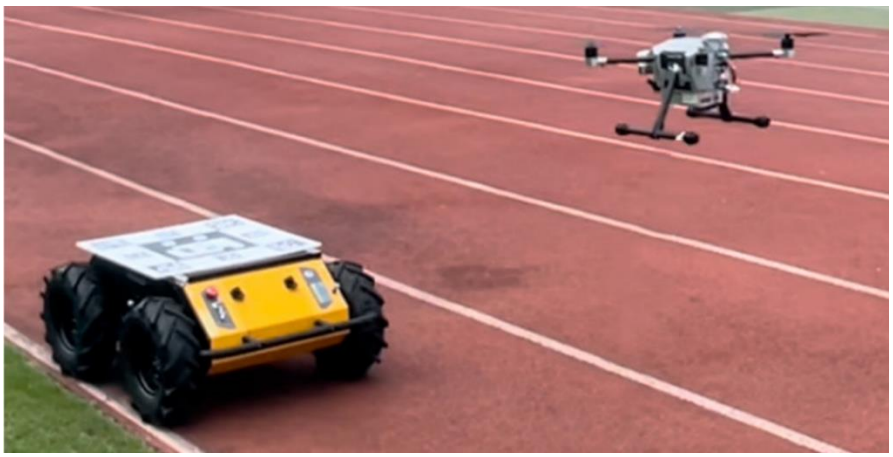
---

[1] helipad

گوشه‌های شناسایی‌شده را به منظور حل مسئله Perspective-n-Points (PnP) مورد استفاده قرار دهند. نتیجه این فرآیند، تخمینی دقیق از موقعیت سکوی فرود نسبت به کوادروتور است.

با این حال، یکی از محدودیت‌های عمده این پژوهش، استفاده از یک نوع خاص از سکوی فرود است که به منظور سهولت در تشخیص، طراحی شده است. این سکوی خاص به‌گونه‌ای طراحی شده که توسط الگوریتم‌های بینایی ماشین به سادگی شناسایی شود. هرچند این رویکرد در محیط‌های آزمایشگاهی نتایج قابل قبولی ارائه می‌دهد، اما از دیدگاه قابلیت تعمیم و کاربرد در پروژه‌های واقعی و در مقیاس وسیع‌تر، محدودیت‌هایی ایجاد می‌کند. در دنیای واقعی، سطح فرود ممکن است دارای ویژگی‌های متنوع و غیرقابل پیش‌بینی باشد، و وابستگی به یک سکوی فرود خاص می‌تواند قابلیت انعطاف‌پذیری و تعمیم‌پذیری سیستم را کاهش دهد.

در پژوهشی دیگر[6]، از کتابخانه ArUco برای شناسایی و تعیین نقطه فرود ربات‌های پرنده استفاده شده است. کتابخانه ArUco به منظور شناسایی و ردیابی مارکرهای چهارگوش[1] طراحی شده و از تکنیک‌های کلاسیک پردازش تصویر برای انجام این کار بهره می‌برد. روش‌های مورد استفاده در ArUco شامل آستانه‌گذاری[2]، تشخیص لبه و استخراج ویژگی‌های هندسی است که به پردازش ساده و مستقیم بر روی پیکسل‌های تصویر متکی هستند. یکی از ویژگی‌های بارز این رویکرد، سرعت بالای شناسایی و ردیابی مارکرها است که آن را به گزینه‌ای مناسب برای کاربردهای بلادرنگ تبدیل می‌کند.



شکل۲-۱ تصویر ربات پرنده که مارکر چهارگوش روی وسیله متحرک را دنبال می‌کند [6]

این سیستم به دلیل نیاز به پردازش‌های سبک و بدون نیاز به محاسبات پیچیده یا استفاده از مدل‌های یادگیری عمیق، از سرعت و کارایی بالایی برخوردار است. این ویژگی، آن را به ابزاری مؤثر در زمینه رباتیک و کنترل ربات‌های پرنده تبدیل می‌کند،

---

[1] squared fiducial markers
[2] thresholding

به‌ویژه در شرایطی که زمان پاسخ‌دهی سریع و دقت بالا مورد نیاز است. علاوه بر این، الگوریتم‌های کنترلی مورد استفاده در این پژوهش به خوبی طراحی شده و عملکردی قابل اعتماد در هدایت و کنترل ربات‌ها به نمایش گذاشته‌اند.

با این حال، یکی از محدودیت‌های عمده استفاده از ArUco، مشابه پژوهش‌های قبلی، مسئله عدم تعمیم‌پذیری است. این سیستم به مارکرهای خاصی وابسته است که برای شناسایی و ردیابی طراحی شده‌اند. در نتیجه، عملکرد آن محدود به محیط‌هایی است که این مارکرها به‌طور خاص نصب شده‌اند و در دنیای واقعی که ممکن است سطح فرود دارای ویژگی‌های غیرمنتظره و متنوع باشد، کارایی آن کاهش می‌یابد. این وابستگی به یک نوع خاص از مارکرها، قابلیت استفاده از این سیستم را در پروژه‌های وسیع‌تر و شرایط واقعی محدود می‌کند.

این مشکل همچنین در مقالات دیگر نیز به چشم می‌خورد.[8][7]

۳-۲ نتیجه گیری و جمع بندی

در تحلیل کارهای مشابه در زمینه فرود ربات‌های پرنده بدون سرنشین بر روی سطوح متحرک، مشخص می‌شود که این پژوهش‌ها به طور عمده از دو رویکرد مختلف استفاده کرده‌اند: الگوریتم‌های پیشرفته بینایی ماشین و تکنیک‌های پردازش تصویر. پژوهش گروه UZH Robotics and Perception به دلیل استفاده از الگوریتم‌های بینایی ماشین و RANSAC، توانسته است فرود دقیق بر روی سطوح متحرک را به خوبی انجام دهد، اما استفاده از سکوی فرود خاص محدودیت‌هایی در قابلیت تعمیم‌پذیری به پروژه‌های واقعی ایجاد کرده است. به همین ترتیب، پژوهش مبتنی بر کتابخانه ArUco به دلیل سرعت بالا و پردازش‌های ساده، به گزینه‌ای مناسب برای شناسایی بلادرنگ تبدیل شده است، اما وابستگی به مارکرهای خاص همچنان محدودیت‌هایی در انعطاف‌پذیری و تعمیم‌پذیری این سیستم‌ها در شرایط واقعی به وجود آورده است. به طور کلی، این کارها نقاط قوت و ضعف مشابهی را نشان می‌دهند که باید در پروژه‌های آینده مورد توجه قرار گیرند تا قابلیت تعمیم‌پذیری و کاربرد در محیط‌های متنوع بهبود یابد.

فصل سوم: روش پیشنهادی و نتیجه‌گیری

۳-۱ مقدمه

در فصل سوم، به ارائه روش پیشنهادی و تحلیل نتایج حاصل از آن خواهیم پرداخت. با توجه به تحلیل و بررسی کارهای مشابه، هدف ما این است که قابلیت تشخیص سکوی‌های فرود را از فواصل دورتر بهبود بخشیم و نزدیک‌تر به شرایط واقعی، عملیاتی نماییم. برای دستیابی به این هدف، ما از شبکه‌های عصبی کانولوشنال استفاده خواهیم کرد. این روش به ما این امکان را می‌دهد که با بهره‌گیری از توانایی‌های پردازش تصویر پیشرفته، عملکرد سیستم را در شناسایی و تشخیص دقیق سکوی فرود در محیط‌های پیچیده و در فواصل مختلف، ارتقاء دهیم. در این فصل، به تفصیل به توضیح ساختار و پیاده‌سازی روش پیشنهادی، و همچنین به تحلیل نتایج به‌دست‌آمده خواهیم پرداخت.

۳-۲ ساختار روش پیشنهادی

ساختار روش پیشنهادی به این صورت است که زبان برنامه‌نویسی پایتون[۱] به عنوان واسط اصلی بین محیط شبیه‌سازی، مدل‌های شبکه عصبی، و کتابخانه‌های پردازش تصویر عمل خواهد کرد. با استفاده از Python، ارتباط و هماهنگی بین این اجزا به‌طور مؤثر برقرار می‌شود و فرآیند پردازش داده‌ها و کنترل ربات بهینه‌سازی خواهد شد. به‌طور خاص، ساختار روش پیشنهادی شامل چهار بخش اصلی زیر است:

- **محیط شبیه‌سازی ویباتز[9][۲]**

ویباتز یک پلتفرم شبیه‌سازی رباتیک پیشرفته است که امکان شبیه‌سازی محیط‌های پیچیده و ربات‌های مختلف را فراهم می‌کند. این محیط به ویژه برای توسعه و آزمایش الگوریتم‌های رباتیک بسیار مناسب است، زیرا به کاربران اجازه می‌دهد تا رفتار ربات‌ها و تعامل آن‌ها با محیط‌های شبیه‌سازی شده را تحلیل کنند. با استفاده از ویباتز ، می‌توانیم محیط‌های شبیه‌سازی شده با ویژگی‌های مختلف، مانند سطوح متحرک و موانع مختلف، ایجاد کنیم و عملکرد ربات‌های پرنده را در این شرایط مختلف مورد بررسی قرار دهیم. ویباتز همچنین ابزارهایی برای طراحی و پیاده‌سازی سنسورها و عملگرهای مختلف فراهم می‌آورد که برای شبیه‌سازی دقیق نیاز است.

---

۱ Python
۲ Webots

- **زبان برنامه‌نویسی پایتون[1]**

پایتون به دلیل ویژگی‌های خود، زبان برنامه‌نویسی انتخابی برای توسعه الگوریتم‌ها و کنترل ربات است. این زبان به خاطر سادگی و خوانایی کد، توانایی‌های علمی و مهندسی، و پشتیبانی گسترده از کتابخانه‌های مختلف، برای پیاده‌سازی الگوریتم‌های پیچیده بسیار مناسب است. در این پروژه، پایتون برای ارتباط با محیط شبیه‌سازی ویباتز ، پردازش داده‌های دریافتی از سنسورها، و پیاده‌سازی الگوریتم‌های بینایی ماشین و کنترل استفاده خواهد شد. پایتون همچنین با دارا بودن کتابخانه‌های متنوع مانند NumPy، OpenCV، و TensorFlow، ابزارهای لازم برای پردازش تصویر و یادگیری ماشین را فراهم می‌آورد.

- **شبکه YOLOv5**

شبکه YOLOv5 یکی از مدل‌های پیشرفته و سریع برای تشخیص اشیاء در تصاویر است. این شبکه با استفاده از ساختارهای پیچیده و بهینه، قادر به شناسایی و مکان‌یابی اشیاء مختلف در تصاویر با سرعت و دقت بالا است. نسخه Small این مدل، برای کاربردهای بلادرنگ و پردازش سریع طراحی شده است، که به ویژه برای شناسایی و تعیین موقعیت باند فرود در ارتفاع بالا مفید است. این مدل با آموزش بر روی تصاویر مختلف از انواع باندهای فرود، قادر به شناسایی دقیق باندهای فرود مختلف و در شرایط متنوع است. با استفاده از YOLOv5، می‌توانیم موقعیت دقیق باند فرود را از فاصله‌های دورتر شناسایی کنیم و ربات پرنده را به سمت آن هدایت کنیم.

- **تکنیک‌های پردازش تصویر سبک**

هنگامی که ربات به نزدیکی باند فرود می‌رسد، استفاده از الگوریتم‌های پردازش تصویر سبک برای بهبود دقت تشخیص و سرعت پردازش ضروری است. این تکنیک‌ها شامل روش‌هایی مانند شکل‌یابی، لبه‌یابی، و تحلیل رنگ هستند که به صورت مستقیم و ساده بر روی پیکسل‌های تصویر کار می‌کنند. این تکنیک‌ها به دلیل نیاز کمتر به محاسبات پیچیده و پردازش سریع، برای شرایط بلادرنگ مناسب هستند و می‌توانند دقت نهایی سیستم را در مراحل پایانی فرود افزایش دهند.

---

[1] Python

با استفاده از ترکیب این ابزارها و تکنیک‌ها، هدف این است که یک سیستم فرود دقیق و مؤثر برای ربات‌های پرنده طراحی و پیاده‌سازی کنیم که قابلیت شبیه‌سازی در محیط‌های مختلف و تطبیق با شرایط واقعی را فراهم می‌آورد.

۳-۳ پیاده‌سازی روش پیشنهادی

۳-۳-۱ محیط شبیه‌سازی و پیکربندی دیوایس‌ها

در این پروژه، از محیط شبیه‌سازی ویباتز برای شبیه‌سازی و آزمایش سیستم استفاده می‌کنیم. برای این منظور، مدل ربات پرنده Mavic 2 Pro را که دارای چهار پره و دوربینی با قابلیت چرخش در محور عمودی است، به محیط شبیه‌سازی اضافه می‌کنیم. Mavic 2 دارای ویژگی‌های پیشرفته‌ای از جمله تثبیت‌کننده تصویر و حسگرهای مختلف است که به ما کمک می‌کند تا نتایج شبیه‌سازی دقیقی به‌دست آوریم.

شکل ۳-۱تصویر ربات پرنده mavic

ویباتز برای mavic یک محیط از پیش ساخته شده دارد که شامل عناصر مختلفی مانند درختان، خانه و ماشین است که به‌طور واقعی‌تری شبیه‌سازی محیط طبیعی را فراهم می‌کند و ما از این محیط استفاده می‌کنیم. به این محیط، ربات Moose اضافه می‌شود که وظیفه جابجایی اجسام را در شبیه‌سازی به عهده دارد. باند فرود در این محیط به‌طور خاص طراحی شده و بر روی Moose قرار می‌گیرد تا تعامل ربات پرنده با سطح فرود به‌طور دقیق شبیه‌سازی شود. بدین ترتیب ربات moose را با سرعت دلخواه حرکت می‌دهیم تا شبیه سازی کامل گردد.

شکل ۳-۲ تصویر ربات moose که باند فرود روی آن قرار گرفته است

در نهایت محیط شبیه سازی نهایی به شکل زیر خواهد بود:

شکل ۳-۳ محیط شبیه‌سازی آماده شده

در این بخش، به بررسی نحوه فعال‌سازی و پیکربندی دستگاه ها و سنسورهای مورد نیاز پروژه با استفاده از کتابخانه کنترلر ویباتز به زبان پایتون می‌پردازیم. این کتابخانه به ما این امکان را می‌دهد که به‌طور مستقیم با سنسورها و عملگرهای ربات تعامل داشته باشیم و داده‌های مورد نیاز را دریافت کنیم.

- دوربین: این کد دوربین ربات را که با نام "camera" شناخته می‌شود، استخراج کرده و آن را با استفاده از تابع enable فعال می‌کند. این دوربین برای اسکن و جمع‌آوری تصاویر از محیط استفاده می‌شود.

- سنسور حرکتی[1]: برای اندازه‌گیری تغییرات زاویه‌ای و شتاب ربات.
- قطب‌نما: برای تعیین جهت و حرکت ربات.
- ژیروسکوپ: برای اندازه‌گیری سرعت زاویه‌ای ربات.
- موتورهای دوربین: موتورهای "camera pitch", "camera roll" و "camera yaw" برای کنترل حرکات دوربین در جهت‌های مختلف به‌کار می‌روند. این موتورها به تنظیم زاویه دوربین و بهبود دقت شناسایی باند فرود کمک می‌کنند.
- موتورهای پره‌ها: شامل موتورهای "front left propeller", "front right propeller", "rear left propeller" و "rear right propeller" که برای کنترل حرکت و مانور ربات پرنده به‌کار می‌روند. این موتورها با تغییر سرعت چرخش خود، حرکت ربات را در جهت‌های مختلف کنترل می‌کنند.

```python
# Get and enable devices.
camera = robot.getDevice("camera")
camera.enable(timestep)
front_left_led = robot.getDevice("front left led")
front_right_led = robot.getDevice("front right led")
imu = robot.getDevice("inertial unit")
imu.enable(timestep)

compass = robot.getDevice("compass")
compass.enable(timestep)
gyro = robot.getDevice("gyro")
gyro.enable(timestep)

camera_roll_motor = robot.getDevice("camera roll")
camera_pitch_motor = robot.getDevice("camera pitch")
camera_yaw_motor = robot.getDevice("camera yaw")  # Added for yaw control

front_left_motor = robot.getDevice("front left propeller")
front_right_motor = robot.getDevice("front right propeller")
rear_left_motor = robot.getDevice("rear left propeller")
rear_right_motor = robot.getDevice("rear right propeller")
motors = [front_left_motor, front_right_motor, rear_left_motor, rear_right_motor]
```

شکل ۳-۴ نحوه فعالسازی سنسور ها و دستگاه ها

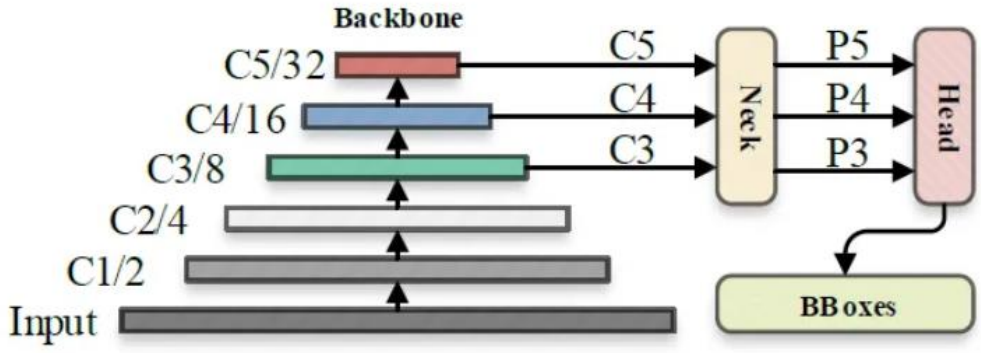
[1] IMU

۳-۳-۲ آموزش و استفاده از شبکه YOLOv5

یکی از چالش‌های اصلی در آموزش مدل‌های یادگیری عمیق برای شناسایی باند فرود، پیدا کردن مجموعه‌ای از تصاویر لیبل‌گذاری شده است که دارای تنوع و جامعیت کافی باشد. مجموعه داده‌های تصویری باید به گونه‌ای باشد که شامل نمونه‌های متنوعی از باندهای فرود با شرایط و ویژگی‌های مختلف باشد تا مدل توانایی تعمیم‌پذیری بالا و عملکرد دقیق در محیط‌های واقعی را داشته باشد.

در این راستا، یکی از منابع ارزشمند که به این مشکل پرداخته، سایت roboflow است.[10] این سایت به‌عنوان یک پایگاه داده تصویری گسترده و عمومی، شامل مجموعه‌ای از تصاویر لیبل‌گذاری شده از باندهای فرود است که می‌توانند برای آموزش مدل‌های یادگیری عمیق مورد استفاده قرار گیرند.

حال به بررسی ساختار مدل خواهیم پرداخت:



شکل ۳-۵ ساختار کلی مدل[11]

- **پیش‌پردازش تصویر و استخراج ویژگی‌ها**[۱]

تصویر ورودی از طریق لایه ورودی پردازش شده و به Backbone برای استخراج ویژگی‌ها ارسال می‌شود. Backbone ویژگی‌های مختلف از تصویر را استخراج کرده و به Feature Fusion Network (Neck) برای ادغام ویژگی‌ها ارسال می‌کند.

---

[۱] Image Preprocessing and Feature Extraction

- **تولید نقشه‌های ویژگی**[۱]

Feature Fusion Network نقشه‌های ویژگی مختلف P3، P4 و P5 را تولید می‌کند که برای شناسایی اشیاء در مقیاس‌های مختلف تصویر استفاده می‌شود.

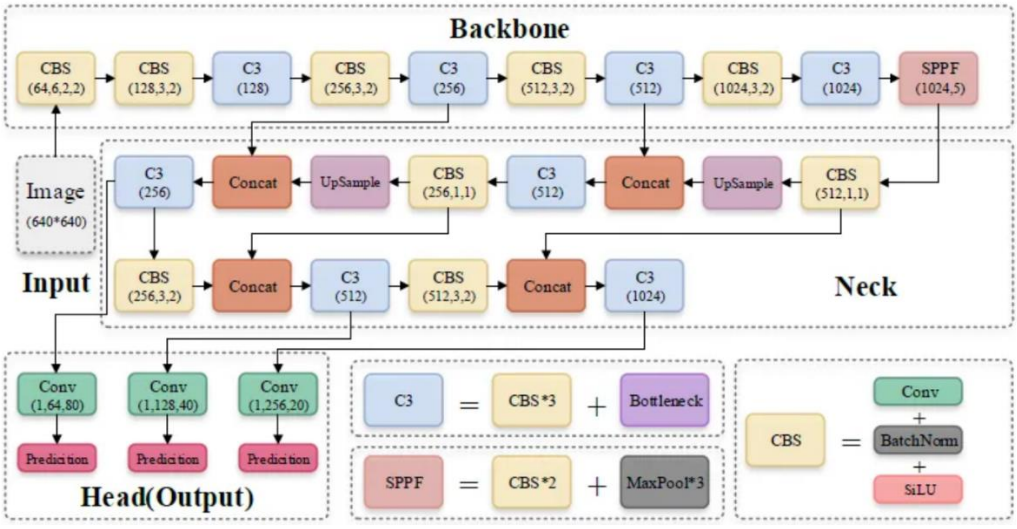
- **پیش‌بینی و پردازش نهایی**[۲]

پس از ارسال نقشه‌های ویژگی به Prediction Head، محاسبات مربوط به اطمینان و رگرسیون جعبه‌های محاط‌کننده برای هر پیکسل انجام می‌شود. این فرآیند با استفاده از anchorهای پیش‌فرض برای پیش‌بینی نهایی ابعاد و موقعیت جعبه‌های محاط‌کننده انجام می‌شود.

- **فیلتر کردن و حذف جعبه‌های محاط‌کننده غیرضروری**[۳]

با تنظیم آستانه‌های مربوطه (confthreshold و objthreshold) و انجام فرآیند جعبه‌های محاط‌کننده غیرضروری، اطلاعات نهایی تشخیص به‌دست آمده و خروجی نهایی مدل تولید می‌شود.

مدل YOLOv5 small شامل چندین نوع لایه است که هر یک نقش خاصی در فرآیند شناسایی اشیاء ایفا می‌کنند. این مدل شامل بخش‌های زیر است:



شکل ۳-۶ جزئیات لایه‌های مدل

---

[۱] Feature Map Generation
[۲] Prediction and Final Processing
[۳] Filtering and Non-Maximum Suppression

- **لایه‌های ورودی[۱]**

ابعاد ورودی[۲] تصاویر به مدل به‌طور معمول دارای ابعاد ۶۴۰ در ۶۴۰ پیکسل هستند. این لایه وظیفه دریافت و آماده‌سازی داده‌های تصویری برای پردازش‌های بعدی را بر عهده دارد.

- **لایه‌های کانولوشنی[۳]**

این لایه‌ها برای استخراج ویژگی‌های پایه از تصاویر استفاده می‌شوند. در YOLOv5 small، این لایه‌ها به‌صورت متوالی برای استخراج ویژگی‌های مقیاس مختلف از تصاویر ورودی طراحی شده‌اند. ساختار این لایه‌ها شامل کانولوشن با فیلترهای مختلف و اندازه‌های هسته متفاوت است که به استخراج ویژگی‌های اولیه از تصویر کمک می‌کند.

- **لایه‌های نرمال‌سازی دسته‌ای[۴] و Leaky ReLU**

نرمال‌سازی دسته‌ای برای نرمال‌سازی و تسریع آموزش مدل استفاده می‌شود. این لایه کمک می‌کند تا توزیع ویژگی‌های لایه‌های قبلی حفظ شده و سرعت آموزش افزایش یابد.

Leaky ReLU تابع فعال‌سازی که در لایه‌های کانولوشن به‌کار می‌رود. این تابع به مدل کمک می‌کند تا از نرمال شدن کامل جلوگیری کند و به عملکرد بهتری برسد.

- **لایه‌های کاهش ابعاد[۵]**

لایه کاهش ابعاد برای کاهش ابعاد ویژگی‌های استخراج‌شده و افزایش قابلیت تعمیم مدل استفاده می‌شود. این لایه به‌طور معمول در لایه‌های مختلف مدل برای کاهش تعداد ویژگی‌ها و حفظ ویژگی‌های اصلی به‌کار می‌رود.

---

۱ Input Layer
۲ Input Dimensions
۳ Convolutional Layers
۴ Batch Normalization
۵ Max Pooling

- **لایه‌های کاهش اندازه[1]**

این لایه‌ها برای کاهش ابعاد تصاویر به‌منظور کاهش تعداد محاسبات و افزایش قابلیت‌های تشخیص ویژگی‌های بزرگ‌تر در تصویر طراحی شده‌اند.

- **لایه‌های Bottleneck**

این لایه ها برای کاهش تعداد پارامترها و محاسبات استفاده می‌شود و همچنین به افزایش کارایی مدل در پردازش ویژگی‌های پیچیده کمک می‌کند. این لایه‌ها معمولاً شامل بلوک‌های چندگانه هستند که به مدل اجازه می‌دهند ویژگی‌های پیچیده‌تری را در مقیاس کوچک‌تر پردازش کنند.

- **لایه‌های شناسایی[2]**

لایه‌های شناسایی شامل سه سر شناسایی[3] برای تشخیص اشیاء در مقیاس‌های مختلف از تصویر است. این لایه‌ها وظیفه پیش‌بینی موقعیت‌ها و کلاس‌های اشیاء را بر عهده دارند. به‌طور خاص، این لایه‌ها سه ویژگی نقشه P3، P4 و P5 را تولید می‌کنند که ابعاد آن‌ها به ترتیب ۸۰×۸۰، ۴۰×۴۰ و ۲۰×۲۰ پیکسل است. این لایه‌ها برای تشخیص اشیاء کوچک، متوسط و بزرگ طراحی شده‌اند.

- **لایه‌های خروجی[4]**

مدل در این لایه پیش‌بینی نهایی را انجام می‌دهد و به‌صورت پیش‌بینی‌های جعبه‌های محاط‌کننده و برچسب‌های کلاس را ارائه می‌دهد. در این مرحله، محاسبات نهایی شامل تعیین موقعیت جعبه‌های محاط‌کننده، برچسب‌های کلاس، و نمرات اطمینان انجام می‌شود.

---

[1] Downsampling
[2] Detection Head
[3] Head
[4] Output Layer

```
Overriding model.yaml nc=80 with nc=1

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16           [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     90880  models.common.C3                        [256, 128, 1, False]
 18                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  1    296448  models.common.C3                        [256, 256, 1, False]
 21                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  1   1182720  models.common.C3                        [512, 512, 1, False]
 24      [17, 20, 23]  1     16182  models.yolo.Detect                      [1, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
Model summary: 214 layers, 7022326 parameters, 7022326 gradients, 15.9 GFLOPs
```

شکل ۳-۷ خلاصه‌ای از لایه‌های مدل

در شکل زیر نتیجه نهایی پردازش مدل روی عکس‌های لیبل گذاری شده را مشاهده می‌کنیم:

```
Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 11/11 [00:04<00:00,  2.31it/s]
                   all        340        358      0.988      0.997      0.995      0.909
```

شکل ۳-۸ نتیجه پردازش مدل

شکل ۳-۹ خروجی مدل بر روی محیط شبیه‌ساز

همانطور که مشخص است در نهایت مدل با دقت بسیار خوبی می‌تواند باندهای فرود متفاوت را به خوبی تشخیص دهد.

همانگونه که در سند های YOLOv5 s نوشته شده و نتایج تست ما نشان می دهد تشخیص هر عکس حدود ۴۰ تا ۵۰ میلی ثانیه زمان می‌برد. پس ما نباید به ازای هر اجرای حلقه تصویر دوربین را به مدل بدهیم و باید به نوعی تاخیر را مدیریت کنیم.

۳-۳-۳ کنترلر

برای کنترل ربات پرنده به‌گونه‌ای که همیشه به سمت جلو و پایین حرکت کند و کادر تشخیص داده شده (باند فرود) در وسط تصویر باقی بماند، کد زیر توسعه یافته است. این کد به‌گونه‌ای طراحی شده که با استفاده از موقعیت باند فرود در تصویر، سرعت و جهت حرکت ربات و همچنین زاویه دوربین را به‌طور پیوسته تنظیم کند.

ربات به‌طور مداوم به سمت جلو و پایین حرکت می‌کند تا به باند فرود نزدیک‌تر شود. این حرکت با تنظیم سرعت موتورهای ربات انجام می‌شود. بر اساس موقعیت باند فرود در تصویر، سرعت حرکت ربات تنظیم می‌شود. اگر باند در وسط تصویر نباشد، سرعت ربات به‌گونه‌ای تنظیم می‌شود که ربات به‌طور مستقیم به سمت باند حرکت کند.

اگر باند فرود از مرکز تصویر خارج شود، زاویه دوربین تنظیم می‌شود تا باند فرود دوباره به مرکز تصویر برگردد. این کار به‌وسیله تنظیم موقعیت دوربین در محورهای عمودی و افقی انجام می‌شود. تنظیمات دوربین باعث می‌شود که باند فرود به‌طور پیوسته در وسط تصویر باقی بماند و به‌این‌ترتیب دقت سیستم شناسایی بهبود یابد.

تصاویر گرفته شده از دوربین به مدل YOLOv5 ارسال می‌شوند تا باند فرود شناسایی شود. مدل YOLOv5، با استفاده از یادگیری عمیق، موقعیت باند فرود را در تصویر تعیین می‌کند. موقعیت باند فرود در تصویر به‌طور دقیق استخراج می‌شود تا برای تنظیم حرکت ربات و زاویه دوربین استفاده شود.

```
# Main control loop
while robot.step(timestep) != -1:
    roll = imu.getRollPitchYaw()[0]
    pitch = imu.getRollPitchYaw()[1]
    roll_acceleration = gyro.getValues()[0]
    pitch_acceleration = gyro.getValues()[1]
    camera_pitch_motor.setPosition(-0.1 * pitch_acceleration)

    roll_disturbance = 0.0
    pitch_disturbance = 0.0
    yaw_disturbance = 0.0
    target_altitude = 0.59
    key = keyboard.getKey()
    if key == Keyboard.UP:
        pitch_disturbance = -2.0
    elif key == Keyboard.DOWN:
        pitch_disturbance = 2.0
    elif key == Keyboard.RIGHT:
        yaw_disturbance = -1.3
    elif key == Keyboard.LEFT:
        yaw_disturbance = 1.3
    elif key == (Keyboard.SHIFT + Keyboard.RIGHT):
        roll_disturbance = -1.0
    elif key == (Keyboard.SHIFT + Keyboard.LEFT):
        roll_disturbance = 1.0
    elif key == (Keyboard.SHIFT + Keyboard.UP):
        target_altitude += 0.5
    elif key == (Keyboard.SHIFT + Keyboard.DOWN):
        target_altitude -= 0.5
    elif key == ord('T') and not thread_running:
        thread.start()
        thread_running = True
    elif key == ord('C'):
        thread.join()
        return 0

    global c_roll_disturbance ,c_pitch_disturbance ,c_camera_pitch_position, c_yaw_disturbance, c_target_altitude, land
    camera_pitch_motor.setPosition(clamp(camera_pitch_position + c_camera_pitch_position,0,1.5))
    roll_input = k_roll_p * clamp(roll, -1.0, 1.0) + roll_acceleration + roll_disturbance + c_roll_disturbance
    pitch_input = k_pitch_p * clamp(pitch, -1.0, 1.0) + pitch_acceleration + pitch_disturbance + c_pitch_disturbance
    yaw_input = yaw_disturbance + c_yaw_disturbance
    vertical_input = k_vertical_p * math.pow(target_altitude + c_target_altitude, 3.0)

    if land:
        front_left_motor_input = 0
        front_right_motor_input = 0
        rear_left_motor_input = 0
        rear_right_motor_input = 0
    else:
        front_left_motor_input = k_vertical_thrust + vertical_input - roll_input + pitch_input - yaw_input
        front_right_motor_input = k_vertical_thrust + vertical_input + roll_input + pitch_input + yaw_input
        rear_left_motor_input = k_vertical_thrust + vertical_input - roll_input - pitch_input + yaw_input
        rear_right_motor_input = k_vertical_thrust + vertical_input + roll_input - pitch_input - yaw_input

    front_left_motor.setVelocity(front_left_motor_input)
    front_right_motor.setVelocity(-front_right_motor_input)
    rear_left_motor.setVelocity(-rear_left_motor_input)
    rear_right_motor.setVelocity(rear_right_motor_input)
```

شکل ۳-۱۰ کد کنترلر

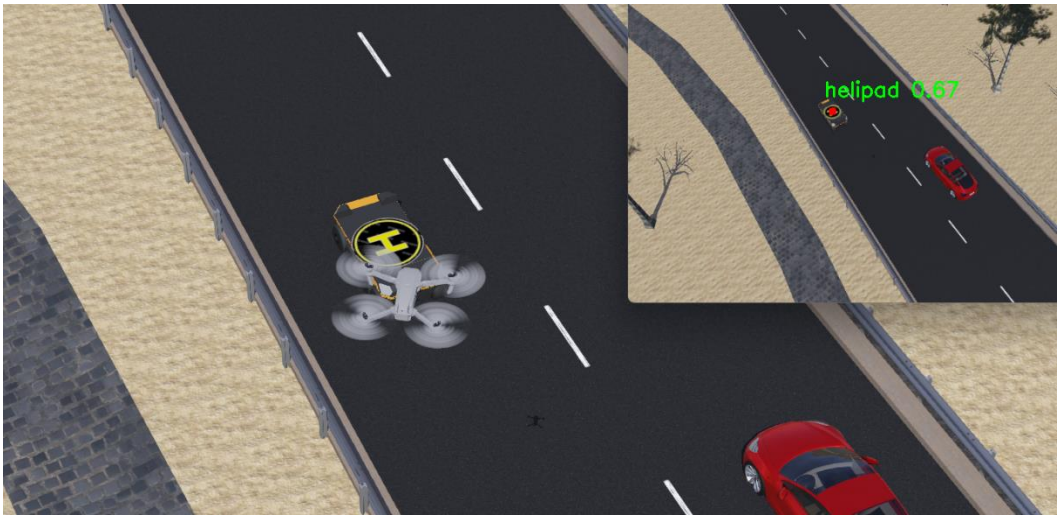
در ابتدای کد، امکان کنترل دستی ربات به وسیله کیبورد فراهم شده است تا امکان تست حالت‌های مختلف برای کاربر مهیا شود. سپس متغیرهای سراسری[1] را مشاهده می‌کنیم که برای کنترل موتورهای ربات استفاده شده‌اند. این متغیرها در توابع پردازش تصویر، با توجه به موقعیت سکوی فرود مقداردهی می‌شوند. در نهایت، اگر ربات در حالت فرود نباشد (که این وضعیت نیز در توابع پردازشی مشخص می‌شود)، مقادیر محاسبه‌شده برای موتورها به آن‌ها اعمال می‌شود.

توابع پردازش تصویر که به صورت یک رشته[2] جداگانه اجرا میشوند همانطور که گفته شد شامل دو مرحله هستند. در مرحله اول ابتدا تا زمانی که مدل ما یک سکوی فرود را شناسایی نکرده باشد به جست و جو در محیط اطراف خودش می پردازد. سپس با پیدا کردن سکوی فرود در تصویر با بیشترین احتمال شروع به حرکت به سمت آن می کند:

```python
def image_processing():
    global c_roll_disturbance, c_pitch_disturbance, c_camera_pitch_position, c_yaw_disturbance, c_target_altitude, land
    first_step = True
    c_camera_pitch_position = 0.8  + camera_pitch_position

    while True:
        sleep(1)
        image = camera.getImageArray()
        if image:
            image_np = np.array(image, dtype=np.uint8)
            img_rgb = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
            img_rgb = cv2.rotate(img_rgb, cv2.ROTATE_90_CLOCKWISE)
            img_rgb = cv2.resize(img_rgb, (length, width))
            img_rgb = cv2.flip(img_rgb, 1)
            results = model(img_rgb)

            highest_confidence_result = None
            max_confidence = 0
            for result in results.pandas().xyxy[0].to_dict(orient='records'):
                confidence = result['confidence']
                if confidence > max_confidence:
                    max_confidence = confidence
                    highest_confidence_result = result

            if highest_confidence_result:
                bbox = highest_confidence_result['xmin'], highest_confidence_result['ymin'], highest_confidence_result['xmax'], highest_confidence_result['ymax']
                label = highest_confidence_result['name']
                confidence = highest_confidence_result['confidence']

                center_x = int((bbox[0] + bbox[2]) / 2)
                center_y = int((bbox[1] + bbox[3]) / 2)
                cv2.circle(img_rgb, (center_x, center_y), radius=5, color=(0, 0, 255), thickness=-1)

                cv2.putText(img_rgb, f'{label} {confidence:.2f}', (int(bbox[0]), int(bbox[1] - 10)),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
```

شکل ۳-۱۱ بخش تشخیص سکوی فرود توسط مدل

[1] Global
[2] Thread



شکل ۳-۱۲ نمونه ای از تشخیص سکوی فرود توسط مدل

سپس تا زمانی که زاویه دوربین به‌طور کامل عمود نشده است، بدون کاهش ارتفاع، فقط به سمت هدف حرکت می‌کنیم. با عمود شدن زاویه دوربین، شروع به کاهش ارتفاع تا ۱۰ متر می‌کنیم. در این موقعیت، چون از درست بودن هدف اطمینان یافته‌ایم، مرحله اول به پایان می‌رسد و می‌توانیم با اطمینان بیشتر و با سرعت بالاتر از توابع پردازشی در مرحله دوم استفاده کنیم:

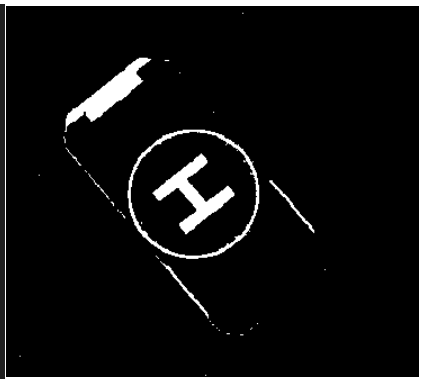
```
if first_step:
    c_yaw_disturbance = clamp(-(center_x - length/2) * 0.01, -0.2, 0.2)

    if center_y < 0.65 * width:
        c_pitch_disturbance = -2
    else:
        c_pitch_disturbance = 2.1

    if center_y > 0.6 * width and c_camera_pitch_position + camera_pitch_position  < 1.36:
        c_camera_pitch_position += 0.15
        print("camera_pitch_position change", c_camera_pitch_position)

    if c_camera_pitch_position + camera_pitch_position >= 1.36:
        first_step = False
else:
    position = gps.getValues()
    height = position[2]
    print(height)
    if height <= 10:
        second_step()
        return
    else:
        c_target_altitude = - 1

    c_yaw_disturbance = clamp(-(center_x - length/2) * 0.01, -0.2, 0.2)
    c_pitch_disturbance = clamp((center_y - width/2) * 0.02, -1.5, 1.5)
```

شکل۳-۱۳ کنترلر مرحله اول

برای بهینه‌سازی فرود ربات پرنده و افزایش دقت در شناسایی و فرود، در مراحل پایانی نزدیک شدن به باند فرود، می‌توانیم از تکنیک‌های پردازش تصویر سبک‌تر مانند لبه‌یابی (Edge Detection) و تکنیک‌های دیگر برای تشخیص دقیق‌تر استفاده کنیم. این روش‌ها به ما کمک می‌کنند تا هنگام نزدیک شدن به باند فرود، فرآیند شناسایی و فرود را به صورت real-time و با دقت بالا انجام دهیم.

در مرحله دوم می‌توان از توابع مختلف استفاده کرد. در فاز اول، تصویر ورودی را به تابع create_mask می‌دهیم تا نویزها و جزئیات اضافی حذف شده و تصویر به یک تصویر باینری تبدیل شود:

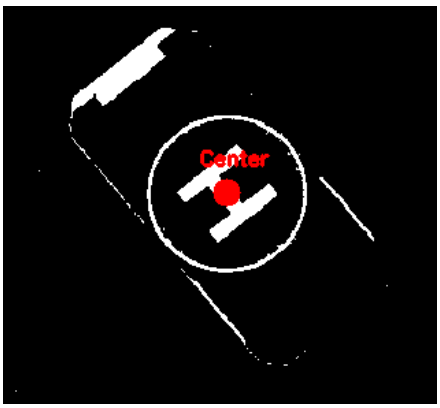
```python
def create_mask(image):
    image = cv2.cvtColor(image,cv2.COLOR_BGR2HLS)
    lower = np.uint8([0, 200, 0])
    upper = np.uint8([255, 255, 255])
    white_mask = cv2.inRange(image, lower, upper)

    lower = np.uint8([10, 0,   100])
    upper = np.uint8([40, 255, 255])
    yellow_mask = cv2.inRange(image, lower, upper)
    mask = cv2.bitwise_or(white_mask, yellow_mask)

    # cv2.imshow("mask", mask)
    # cv2.waitKey(1)
    return mask
```



شکل ۳-۱۴ تابع create_mask و خروجی آن

در گام بعد، تصویر خروجی تابع بالا را به تابع find_H می‌دهیم تا با استفاده از تکنیک بیشترین مساحت خطوط بسته[۱]، نقطه مرکزی سکوی فرود را پیدا کند:

```python
def find_H(image):
    _, binary = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if contours:
        largest_contour = max(contours, key=cv2.contourArea)

        M = cv2.moments(largest_contour)
        if M["m00"] != 0:
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
        else:
            cX, cY = 0, 0

        output_image = cv2.cvtColor(binary, cv2.COLOR_GRAY2BGR)
        cv2.circle(output_image, (cX, cY), 10, (0, 0, 255), -1)
        cv2.putText(output_image, "Center", (cX - 20, cY - 20), cv2.FONT_HERSHEY_SIMPLEX

        cv2.imshow("find_H", output_image)
        cv2.waitKey(1)
        return(cX, cY)
```

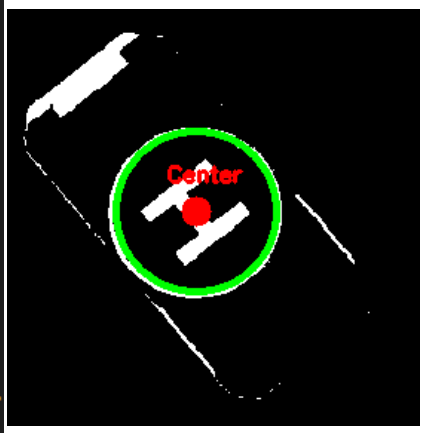


شکل ۳-۱۵ تابع find_H و خروجی آن

[۱] contourArea

همچنین، اگر اطمینان داشته باشیم که سکوی فرود به شکل دایره است، می‌توانیم از این ویژگی برای استخراج مرکز آن از تصویر استفاده کنیم. البته این روش نسبت به روش قبلی عمومیت و جامعیت کمتری دارد، به همین دلیل از تابع قبلی در کد استفاده می‌شود:

```python
def find_circle(gray):
    blurred = cv2.GaussianBlur(gray, (9, 9), 2)

    circles = cv2.HoughCircles(
        blurred,
        cv2.HOUGH_GRADIENT,
        dp=1.2,  # Inverse ratio of the accumulator resolution to the image resolution
        minDist=500,  # Minimum distance between detected centers
        param1=70,  # Higher threshold for Canny edge detector
        param2=50,  # Accumulator threshold for circle detection
        minRadius=10,  # Minimum radius to be detected
        maxRadius=0  # Maximum radius to be detected (0 means no max limit)
    )
    output_image = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)

    if circles is not None:
        circles = np.round(circles[0, :]).astype("int")
        for (x, y, r) in circles:
            cv2.circle(output_image, (x, y), r, (0, 255, 0), 4)
            cv2.circle(output_image, (x, y), 10, (0, 0, 255), -1)
            cv2.putText(output_image, "Center", (x - 20, y - 20), cv2.FONT_HERSHEY_SIMPLEX,

    cv2.imshow("find_circle", output_image)
    cv2.waitKey(1)
```



شکل ۳-۱۶ تابع find_circle و خروجی آن

بنابراین، در مرحله دوم با استفاده از توابع create_mask و find_H مرکز سکوی فرود را پیدا می‌کنیم و مانند مرحله اول، متغیرهای کنترلی سراسری را مقداردهی می‌کنیم. به این صورت که تا ارتفاع ۱.۵ متری (با توجه به ارتفاع سکوی فرود) به صورت کنترل‌شده ارتفاع را کاهش می‌دهیم و سپس با خاموش کردن موتورها فرود انجام می‌شود و کار این رشته به پایان می‌رسد:

```python
def second_step():
    global c_roll_disturbance, c_pitch_disturbance, c_camera_pitch_position, c_yaw_disturbance, c_target_altitude, land

    while True:
        sleep(0.1)
        image = camera.getImageArray()
        if image:
            image_np = np.array(image, dtype=np.uint8)
            img_rgb = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
            img_rgb = cv2.rotate(img_rgb, cv2.ROTATE_90_CLOCKWISE)
            img_rgb = cv2.resize(img_rgb, (length, width))
            img_rgb = cv2.flip(img_rgb, 1)

            mask = create_mask(img_rgb)

            center_x , center_y = find_H(mask)
            # find_circle(mask)

            position = gps.getValues()
            height = position[2]
            print(height)

            c_target_altitude = - 0.3
            c_roll_disturbance = clamp(-(center_x - length/2) * 0.1, -1,1)
            c_pitch_disturbance = clamp((center_y - width/2) * 0.02 , -1.5, 1.5)

            if center_x < length * 0.35 or center_x > length * 0.65:
                c_yaw_disturbance = clamp(-(center_x - length/2) * 0.001, -0.4, 0.4)

            if height < 1.5:
                c_roll_disturbance = 0
                c_pitch_disturbance = 0
                land = True
                return
```

شکل ۳-۱۶ کنترلر مرحله دوم

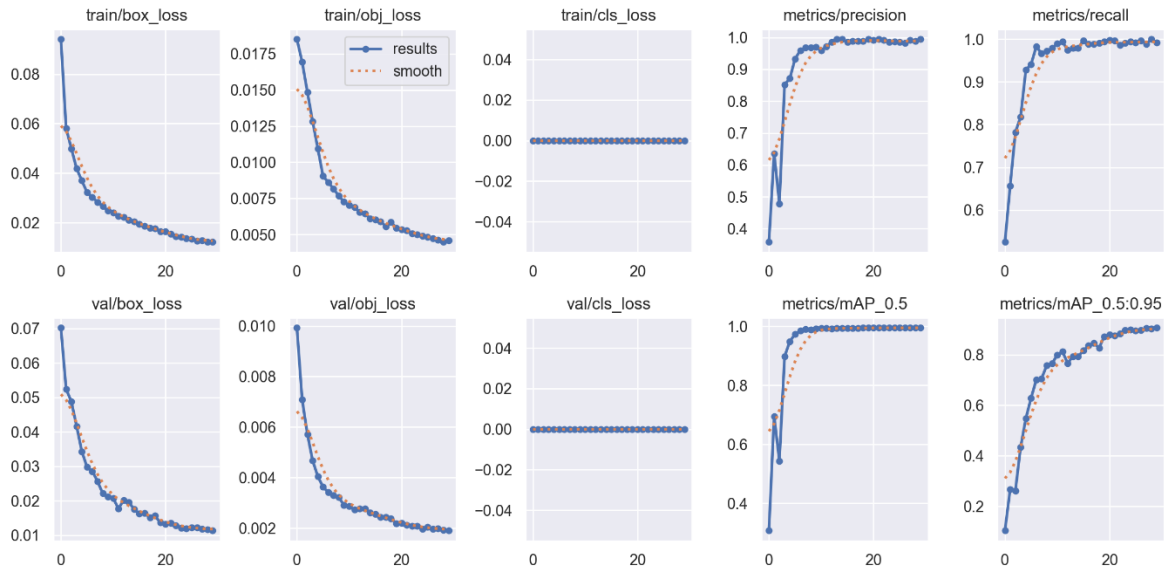
برای اینکه ربات Moose نیز حرکت مورد نظر ما را انجام دهد، نیاز به یک کنترلر داریم که کد آن در زیر آورده شده است:

```python
from controller import Robot

robot = Robot()

timestep = int(robot.getBasicTimeStep())

motor_names = [
    "left motor 1", "left motor 2", "left motor 3", "left motor 4",
    "right motor 1", "right motor 2", "right motor 3", "right motor 4"
]

motors = [robot.getDevice(name) for name in motor_names]

for motor in motors:
    motor.setPosition(float('inf'))

def set_velocity(left_speed, right_speed):
    for i in range(4):
        motors[i].setVelocity(left_speed)  # Left motors
    for i in range(4, 8):
        motors[i].setVelocity(right_speed)  # Right motors

forward_speed = 1.5  # Adjust the speed as necessary
backward_speed = -1.5  # Negative for backward movement
move_duration = 20000  # Adjust this for how long to move in each direction

# Main loop
while robot.step(timestep) != -1:
    for _ in range(move_duration):
        set_velocity(forward_speed, forward_speed)
        if robot.step(timestep) == -1:
            break
    set_velocity(0.0, 0.0)

    # Move backward
    for _ in range(move_duration):
        set_velocity(backward_speed, backward_speed)
        if robot.step(timestep) == -1:
            break
    set_velocity(0.0, 0.0)
```

شکل ۳-۱۷ کنترلر ربات moose

۳-۴ ارزیابی

نمودارهایی که در ادامه آمده اند، نتایج آموزش مدل YOLOv5s بر روی مجموعه‌ای از تصاویر سکوهای فرود را نشان می‌دهند. تحلیل و ارزیابی این نتایج نیازمند بررسی دقیق هر یک از معیارهای آموزش و اعتبارسنجی است که در این نمودارها به تصویر کشیده شده‌اند. در ادامه به تحلیل هر کدام از این نمودارها می‌پردازیم:



شکل ۳-۱۸ نتایج آماری مدل

- **Train/Box_Loss و Val/Box_Loss:**

Train/Box_Loss: این نمودار نشان‌دهنده خطای مربوط به موقعیت و اندازه جعبه‌های محدودکننده[۱] در داده‌های آموزشی است. این معیار اهمیت زیادی دارد چرا که دقت مدل در تشخیص موقعیت دقیق سکوهای فرود را نشان می‌دهد. کاهش مداوم این خطا در طول اپوک‌ها نشان می‌دهد که مدل به تدریج در حال یادگیری است و دقت خود را در پیش‌بینی ها بهبود می‌بخشد.

Val/Box_Loss: این نمودار نیز مشابه Train/Box_Loss است، اما برای داده‌های اعتبارسنجی. کاهش این خطا نشان‌دهنده این است که مدل توانسته است ویژگی‌های سکوهای فرود را به خوبی یاد بگیرد و این دانش را به داده‌های جدید و ندیده تعمیم دهد.

---

[۱] bounding boxes

کاهش مداوم هر دو خطا نشان‌دهنده این است که مدل به درستی آموزش دیده و دچار بیش برازش[1] نشده است، چرا که خطاهای اعتبارسنجی نیز همراه با خطاهای آموزش کاهش یافته‌اند.

- **Train/Obj_Loss و Val/Obj_Loss:**

Train/Obj_Loss: این نمودار بیانگر خطای مربوط به تشخیص اشیا در داده‌های آموزشی است. این خطا نشان می‌دهد که مدل چقدر در شناسایی وجود یک شیء (در این مورد سکوی فرود) در تصویر موفق بوده است.

Val/Obj_Loss: این نمودار خطای مربوط به داده‌های اعتبارسنجی را نشان می‌دهد. کاهش مشابه در هر دو نمودار بیانگر این است که مدل در تشخیص وجود یا عدم وجود سکوهای فرود در داده‌های جدید و دیده‌نشده نیز موفق بوده است.

کاهش مداوم این خطاها نشان می‌دهد که مدل به خوبی توانسته است وجود سکوهای فرود را در تصاویر تشخیص دهد و عملکرد آن در داده‌های ندیده نیز مناسب است.

- **Train/Cls_Loss و Val/Cls_Loss:**

Train/Cls_Loss و Val/Cls_Loss: هر دو نمودار نشان می‌دهند که خطای کلاس‌بندی در تمام مراحل آموزش برابر با صفر بوده است. این به این معناست که مدل نیازی به تفکیک بین چندین کلاس مختلف نداشته است (چرا که همه سکوهای فرود به یک کلاس تعلق دارند).

عدم تغییر این خطا نشان‌دهنده این است تعداد کلاس‌های مدنظر تنها یک کلاس بوده است.

- **Metrics/Precision:**

Precision: این نمودار نشان‌دهنده دقت مدل در شناسایی صحیح سکوهای فرود است. افزایش تدریجی دقت مدل در طول زمان نشان می‌دهد که مدل به مرور در تشخیص صحیح اشیا بهتر عمل کرده است و تعداد کمی از پیش‌بینی‌های مثبت آن اشتباه بوده است.

دقت بالای مدل بیانگر این است که بیشتر اشیا که به عنوان سکوی فرود تشخیص داده شده‌اند، واقعاً سکوی فرود بوده‌اند و مدل تعداد کمی خطای مثبت کاذب[2] داشته است.

---

[1] overfitting
[2] false positives

- **Metrics/Recall**:

Recall: این نمودار بیانگر میزان فراخوانی مدل است که نشان می‌دهد چه تعداد از سکوهای فرود موجود در تصویر به درستی توسط مدل تشخیص داده شده‌اند. افزایش قابل ملاحظه‌ای در recall در اوایل مراحل آموزش دیده می‌شود که نشان‌دهنده بهبود سریع مدل در شناسایی تمامی سکوهای فرود موجود در تصویر است.

افزایش recall نشان می‌دهد که مدل توانسته است تقریباً تمامی سکوهای فرود موجود در تصاویر را شناسایی کند، بدون این که تعداد زیادی شیء واقعی را از دست بدهد[1].

- **Metrics/mAP_0.5 و Metrics/mAP_0.5:0.95**:

میانگین دقت متوسط[2]: این معیار یک شاخص جامع برای ارزیابی عملکرد مدل در تشخیص اشیا در مقیاس‌های مختلف است. mAP_0.5 به معنای میانگین دقت در آستانه[3] برابر با ۰.۵ است، در حالی که mAP_0.5:0.95 میانگین دقت در آستانه‌های مختلف بین ۰.۵ تا ۰.۹۵ را اندازه‌گیری می‌کند.

mAP_0.5: روند افزایشی و بالای این معیار نشان‌دهنده عملکرد خوب مدل در شناسایی دقیق سکوهای فرود است.

mAP_0.5:0.95: این معیار نیز روند افزایشی مشابهی را نشان می‌دهد، اما این بار با در نظر گرفتن آستانه‌های سخت‌گیرانه‌تر.

افزایش مداوم mAP در هر دو آستانه نشان‌دهنده عملکرد قابل‌اطمینان مدل در شناسایی دقیق و صحیح سکوهای فرود است، حتی در شرایط سخت‌گیرانه‌تر.

این نمودارها به وضوح نشان می‌دهند که مدل YOLOv5s به خوبی بر روی مجموعه داده سکوهای فرود آموزش دیده است. کاهش خطاهای مختلف و افزایش معیارهای دقت و فراخوانی نشان‌دهنده یادگیری صحیح و توانایی تعمیم مدل به داده‌های جدید است. به طور کلی، این مدل عملکردی قوی در تشخیص و شناسایی سکوهای فرود دارد و نتایج آن قابل‌اعتماد و امیدوارکننده است.

---

[1] false negatives
[2] Mean Average Precision (mAP)
[3] Intersection over Union

۳-۵ نتایج

در این بخش، نتایج حاصل از ارزیابی روش پیشنهادی به‌طور دقیق بررسی و تحلیل می‌شود. هدف از این تحلیل، سنجش موفقیت‌های پروژه و شناسایی نقاط ضعف آن است. علاوه بر ارائه داده‌های آماری، باید به تفسیر این داده‌ها و دستیابی به اهداف پروژه پرداخته شود.

- **دقت شناسایی باند فرود**

مدل YOLOv5 small به‌خوبی توانسته است باندهای فرود مختلف را شناسایی کند. نتایج نشان می‌دهند که مدل در تشخیص باندهای فرود با دقت بالا عمل کرده است و دقت شناسایی (Precision) و یادآوری (Recall) مدل در حد قابل قبولی قرار دارد. تحلیل نتایج نشان می‌دهد که مدل قادر به تشخیص باندهای فرود با انواع و شرایط نوری مختلف است.

- **زمان پردازش و سرعت مدل**

با وجود دقت بالای مدل در شناسایی باند فرود، در بحث زمان پردازش و سرعت اجرای الگوریتم‌ها، نقاط ضعفی مشاهده شده است. زمان پردازش هر تصویر و اجرای الگوریتم‌ها بر روی داده‌های واقعی بیشتر از حد مطلوب است و نیاز به بهینه‌سازی دارد. این موضوع ممکن است به دلیل پیچیدگی مدل و یا نیاز به بهینه‌سازی کدهای پردازش تصویر باشد.

- **کنترل حرکت ربات و دقت فرود**

کنترلر ربات به‌طور عمومی توانسته است ربات را به سمت باند فرود هدایت کند، اما در عمل مشکلاتی مانند ناپایداری در سرعت حرکت و دقت پایین در تنظیم زاویه دوربین برای حفظ کادر شناسایی‌شده در وسط تصویر مشاهده شده است. این ناپایداری‌ها می‌توانند به دلیل الگوریتم‌های کنترلی فعلی و یا حساسیت‌های محیطی باشند.

۳-۶ جمع‌بندی

هدف شناسایی باندهای فرود با استفاده از مدل YOLOv5 small به‌خوبی محقق شده است. مدل با دقت بالا قادر به شناسایی باندهای مختلف و با شرایط متفاوت بوده است. ولی ربات قادر به حرکت به سمت باند فرود بوده است، اما در بعضی شرایط نیاز به بهبود در دقت و پایداری کنترلر احساس می‌شود و همچنین آموزش مدل با عکس های بیشتر و متنوع تر به بهبود عملکرد کمک شایانی خواهد کرد.

- **نقاط ضعف و نیاز به بهبود**

- سرعت پردازش: بهبود در زمان پردازش تصاویر و اجرای سریع‌تر الگوریتم‌ها برای رسیدن به عملکرد real-time مورد نیاز است.
- کنترلر و هدایت ربات: کنترلر فعلی نیاز به بهبود در تنظیمات سرعت و زاویه دوربین برای حفظ کادر شناسایی‌شده در وسط تصویر دارد. همچنین، پایداری حرکت و دقت فرود باید تقویت شود.

ادعای پژوهشی و علمی ما این است که مدل YOLOv5 small قادر است باندهای فرود را با دقت بالا شناسایی کند و به‌طور کلی عملکرد مناسبی در شناسایی اشیاء در محیط‌های شبیه‌سازی شده دارد. با این حال، برای رسیدن به عملکرد بهینه در شرایط واقعی و real-time، نیاز به بهبود در سرعت پردازش، کنترلر و دقت هدایت ربات وجود دارد.

با توجه به نتایج به‌دست‌آمده، می‌توان گفت که روش پیشنهادی توانسته است در شناسایی باندهای فرود با دقت بالا عمل کند، اما چالش‌هایی در زمینه سرعت پردازش و کنترلر ربات وجود دارد که باید مورد توجه و بهبود قرار گیرد. این نتایج، پایه‌ای برای بهبودهای آینده و توسعه الگوریتم‌های موثرتر برای کنترل ربات‌ها و پردازش تصاویر فراهم می‌آورد.

منابع

[1] International Federation of Robotics, “World Robotics Report 2020,” Available at: https://ifr.org/downloads/press2018/Executive_Summary_WR_2020.pdf, 2020.

[2] R. Mahony, V. Kumar, and P. Corke, “Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 20-32, 2012.

[3] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme, “Visually guided landing of an unmanned aerial vehicle,” *IEEE Transactions on Robotics and Automation*, vol. 19, no. 3, pp. 371-380, 2003.

[4] G. Jocher, “ultralytics/yolov5,” *GitHub*, Aug. 21, 2020. https://github.com/ultralytics/yolov5

[5] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, “Vision-based Autonomous Quadrotor Landing on a Moving Platform.” Accessed: Aug. 24, 2024. [Online]. Available: https://rpg.ifi.uzh.ch/docs/SSRR17_Falanga.pdf

[6] P. Wang, C. Wang, J. Wang, and Max Q.-H. Meng, “Quadrotor Autonomous Landing on Moving Platform,” *Procedia computer science*, vol. 209, pp. 40–49, Jan. 2022, doi: https://doi.org/10.1016/j.procs.2022.10.097.

[7] A. Gautam, M. Singh, P. B. Sujit, and S. Saripalli, “Autonomous Quadcopter Landing on a Moving Target,” *Sensors*, vol. 22, no. 3, p. 1116, Jan. 2022, doi: https://doi.org/10.3390/s22031116.

[8] Azarakhsh Keipour *et al.*, “Visual Servoing Approach to Autonomous ربات‌های پرنده بدون سرنشینLanding on a Moving Vehicle,” *Sensors*, vol. 22, no. 17, pp. 6549–6549, Aug. 2022, doi: https://doi.org/10.3390/s22176549.

[9] “Webots: robot simulator,” *Cyberbotics.com*, 2019. https://cyberbotics.com

[10] dfdfdfdfdfddfdf Object Detection Dataset (v1, 2023-04-08 1:28pm) by new-workspace-mlfvi, “dfdfdfdfdfddfdf Object Detection Dataset (v1, 2023-04-08 1:28pm) by new-workspace-mlfvi,” *Roboflow*, 2023. https://universe.roboflow.com/new-workspace-mlfvi/dfdfdfdfdfddfdf/dataset/1 (accessed Aug. 24, 2024).

[11] J. Solawetz, “YOLOv5 New Version - Improvements And Evaluation,” *Roboflow Blog*, Jun. 29, 2020. https://blog.roboflow.com/yolov5-improvements-and-evaluation/

پیوست

https://github.com/mostafaAtyabi/UAV-landing.git

Autonomous Drone Landing on a Moving Target Based on Artificial Intelligence Approaches in a Simulation Environment

Abstract

This project focuses on the development and evaluation of an automated system designed to guide flying robots to landing pads using simulation in the Webots environment and image processing with the YOLOv5 small model. The primary goal is to create and implement an intelligent system capable of detecting landing pads and autonomously directing the flying robot toward them. Throughout the research, various challenges such as the accuracy of landing pad detection and the stability of robot movement control under real-world conditions were explored.

A key issue in this area is the insufficient precision in identifying landing pads and the accurate control of robot movement in complex environments. Existing models for landing pad detection have limitations in both accuracy and processing speed, necessitating a more efficient solution. This project addresses these issues by employing the YOLOv5 small model for detecting landing pads and utilizing motion control algorithms to navigate the robot.

The proposed method integrates the YOLOv5 small model for detecting landing pads and an intelligent controller for manoeuvring the flying robot. Evaluation results indicate that while the model effectively identifies landing pads, there is room for improvement in processing speed and controller accuracy. The project has successfully met its primary objectives and provides a solid foundation for future advancements in flying robot control and image processing.

Keywords: Flying Robot, Simulation, YOLOv5, Landing Pad Detection, Image Processing, Intelligent Controller

Shahid Beheshti University
Faculty of Computer Science and Engineering

Autonomous Drone Landing on a Moving Target Based on Artificial Intelligence Approaches in a Simulation Environment

By:
Seyed Mostafa Atyabi

A THESIS SUBMITTED
FOR THE DEGREE OF
BACHELOR OF SCIENCE

Supervisor
Dr. Armin Salimi Badr

August 2024